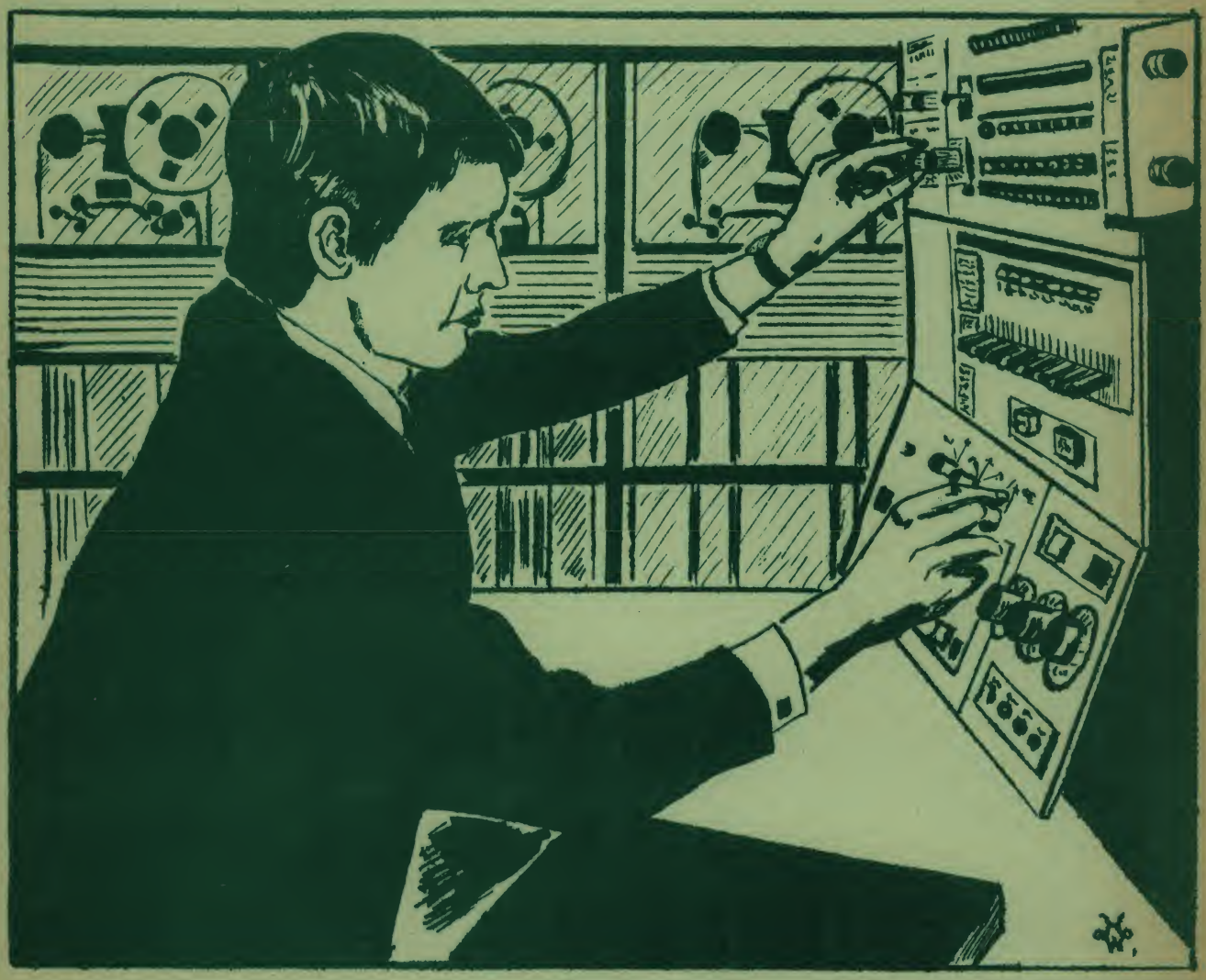
E C S

COMPUTER PROGRAMMEUR



Les 6

SYSTEEMPROGRAMMA'S

Prof. Ir. D.H. Wolbers

Een publikatie van de Stichting Het Nederlands Studiecentrum voor Informatica, Amsterdam.

Ir. D.H. Wolbers

5. INLEESPROGRAMMA'S, COMPILERS e.d.

Reeds vanaf het begin van het gebruik van de " Stored program computer" deed zich het probleem voor om een geschreven en daarna geponst programma op de juiste manier en op de juiste plaats in het geheugen te krijgen.
Daarvoor is op zich een ander programma noodzakelijk,dat reeds in het geheugen van de machine aanwezig is, en meestal bekend staat onder de naam *inleesprogramma* of *invoerprogramma*.
Naarmate de rekenmachines groter en sneller werden, ontstond steeds meer de noodzaak het programmeren zelf te vereenvoudigen. Een machtig hulpmiddel hiervoor vormen in feite de mogelijkheden die in het inleesprogramma gelegd worden.

Een voorbeeld van deze mogelijkheden wordt gegeven door de in deze kursus reeds gebruikte mnemotechnische codering alsmede het gebruik van symbolische adressen.
Moderne inleesprogramma's zijn daarmee uitgegroeid van enkele simpele lees- en opberginstrukties tot grote organisatorische, ingewikkelde programma's.

In het volgende zullen enkele aspekten, die bij het samenstellen van invoerprogramma's optreden, nader worden bezien.

Invoerprogramma's vertonen met vele programma's voor administratieve werkzaamheden het gemeenschappelijk karakter, dat voor het grootste deel met alfanumerieke informatie gemanipuleerd moet worden. Bovendien gebaseerd op deze informatie bestaat het invoerprogramma uit een groot aantal vertakkingen, zodat het programma zelf veelal gevormd wordt door een aaneenschakeling van vele test- en spronginstrukties, waarbij betrekkelijk weinig wordt gerekend. Dit soort bewerkingen kan bovendien vaak versneld en vereenvoudigd worden door in hardware enkele speciale en logische opdrachten aan te brengen.

Dergelijke opdrachten bestaan ook in SERA. Deze worden hierna besproken met enkele voorbeelden van stukjes programma, zoals ze gedeeltelijk ook voorkomen in invoerprogramma's.

Allereerst 2 schuifopdrachten.

BSL n A en B binair schuiven naar links over n plaatsen
BSR n A en B binair schuiven naar rechts over n plaatsen.

Bij deze binaire schuifopdrachten worden A en B als één

geheel van 2 x 48 = 96 bits beschouwd. Het schuiven is bovendien rondgekoppeld, d.w.z. bij BSL worden bits links uit A rechts in B weer ingevoerd en omgekeerd bij BSR. Het adresgedeelte van deze opdrachten geeft het aantal binaire plaatsen aan waarover geschoven wordt. Willen we dus over een hexade-lengte naar links schuiven dan moet dit met BSL 6. Nemen we als voorbeeld, dat in de B accumulator gegeven is een alfanumeriek woord en dat we de binaire waarde van de meest rechtse hexade willen omzetten in een decimaal getal van 2 cijfers.
Dat kan dan met het volgende stukje programma, waarbij voor de verklarende tekst is aangenomen, dat deze hexade een punt voorstelt. De punt is gerepresenteerd door de binaire combinatie 101111 hetgeen decimaal 47 is.

```
HPA * 0        A schoon
BSR   6        101111 links in A
HPB   0        B schoon
BSL   3        101 = 5 rechts in B
BPA   HULP     111 links in A naar hulp
VMG   8        8 x 5 = 40 naar B
HPA   HULP     111 links in A
BSB   HULP     40 naar hulp; B schoon
BSL   3        111 = 7 rechts in B
OPB   HULP     7 + 40 = 47 naar B
```

Door deze schuifopdrachten kunnen we dus zonodig met de individuele bits van een woord gaan rekenen.

LOGISCHE BEWERKINGEN

Het rekenen met afzonderlijke bits noemt men wel logische bewerkingen. Dat betekent niet dat andere bewerkingen onlogisch zouden zijn, maar wel dat deze bewerkingen te maken hebben met de symbolische logica. Men werkt daarbij met tweewaardige elementen zoals "wel" of "niet", "waar" of "onwaar", 0 of 1.
Men kent 2 hoofdbewerkingen namelijk logisch optellen en logisch vermenigvuldigen.

LOGISCH OPTELLEN

Als men spreekt over de logische som van a en b dan schrijft men a+b, of a $\vee$ b of a OF b.
Deze optelling is daarbij gedefinieerd als volgt :

a	b	a + b
0	0	0
1	0	1
0	1	1
1	1	1

Dit betekent : de logische som van a en b is "waar" als tenminste een der beide elementen "waar" is.

LOGISCH VERMENIGVULDIGEN

Het logisch produkt van twee elementen a en b wordt geschreven als axb, of a $\wedge$ b of a EN b, daarbij gedefinieerd als volgt :

a	b	a b
0	0	0
1	0	0
0	1	0
1	1	1

Dit betekent : het logische produkt van a en b is "waar" indien zowel a als b "waar" zijn.

In de symbolische logica werkt men ook veel met de inverse van een element. Men noteert dit door een streepje boven de betrokken naam te plaatsen. Tussen a en a bestaat dan relatie.

a	$\bar{a}$
0	1
1	0

In de spreektaal zegt men voor $\bar{a}$ "niet a".

Er is nu een serie opdrachten waardoor men alle bits van een SERA-woord logisch kan optellen bij of vermenigvuldigen met de overeenkomstige bits van een ander woord.

OFA n logische bewerking OF van (A) met (n) naar A
ENA n logische bewerking EN van ($\bar{A}$) met (n) naar A
OFB n logische bewerking OF van ($\bar{B}$) met (i) naar B
ENB n logische bewerking EN van ($\bar{B}$) met (n) naar B

Met deze opdrachten kan men gemakkelijk woorden "knippen" en samenstellen. We nemen als voorbeeld dat in WOORD1 en WOORD2 twee alfanumerieke woorden beschikbaar zijn. We willen de 4e en 5e hexade van rechts af gerekend van WOORD1 vervangen door resp. de 2e en 3e hexade van rechts van WOORD2. De overige hexaden van WOORD1 alsmede het gehele WOORD2 moeten onaangetast blijven.

```
HPA   0        nullen naar A
HNB   0        enen naar B
BSL   12       12 enen rechts in A
HPB   0        nullen naar B
```

```
BSL   6 in A      nu enen op 2e en 3e hexade-plaats
                  verder nullen
BPA   HULP        dit patroon tijdelijk naar hulp
BSL   12
HNA   ACCU-A      in A nullen op 4e en 5e hexade-
                  plaats, verder enen
ENA   WOORD1      wegknippen van 4e en 5e hexade
                  uit WOORD1
BSA   WOORD1      nullen naar A
HPB   HULP
ENB   WOORD2      uitsnijden van 2e en 3e hexade
                  uit WOORD2
BSL   12          2e en 3e hexade naar 4e en 5e
                  hexadeplaats
OFB   WOORD1      invoegen nieuwe 4e en 5e hexade
                  in WOORD1
BPB   WOORD1
```

Logische opdrachten vindt men eigenlijk bij iedere machine. Dat is niet zo verwonderlijk omdat intern de gehele technische werking feitelijk berust op deze twee EN en OF bewerkingen. Ieder van deze bewerkingen wordt betrekkelijk eenvoudig gerealiseerd door een zgn. poortschakeling.Iedere handeling, die in de machine verricht moet kunnen worden, wordt teruggebracht tot een herhaald aantal malen dergelijke EN en OF manipulaties toepassen. Daarvoor zijn dan een aantal van de genoemde poortschakelingen nodig.

Vooral het besturingsorgaan en het rekenorgaan zijn op deze wijze opgebouwd uit grote aantallen poortschakelingen.

Een ander voorbeeld in de machine is bijv. het selectiemechanisme in het geheugen.

In deze kursus wordt verder niet ingegaan op de technische werking van deze schakelingen. De wijze waarop diverse instrukties echter kunnen worden gerealiseerd is wel van belang. Bij de construktie van een machine is het nu eenmaal onmogelijk te voren alle mogelijke instrukties in te bouwen, die soms voor heel speciale doeleinden ooit nog eens nodig zijn.
Met de logische instrukties kan men dergelijke niet ingebouwde instrukties dan zodanig in de vorm van een subroutine simuleren.
Voor het ontwerpen daarvan is dan vaak nuttig eerst wat met de verschillende logische grootheden te "rekenen" om de meest geschikte vorm te vinden die gebruikt kan worden.

Het rekenen gaat vrijwel gelijk aan de normale algebra, alleen zijn hier soms nog andere mogelijkheden.
Wanneer a, b en c logische waarden voorstellen dan geldt :

Ir. D.H. Wolbers

axb=bxa
a+b=b+a
ax(b+c)=axb+axc
(a+b)+c=a+(b+c)

Nieuw hier is : a+(bxc)=(a+b)x(a+c)
Men kan dergelijke stellingen gemakkelijk bewijzen door eenvoudig alle mogelijkheden na te gaan.
Ook door het gebruik van de "niet" versies ontstaan twee veel gebruikte regels :

$\bar{a}\bar{x}\bar{b} = \bar{a}+\bar{b}$

$\bar{a}\bar{+}\bar{b} = \bar{a}x\bar{b}$

We bewijzen als voorbeeld eens de laatste stelling.

a	b	a+b	$\overline{a+b}$	$\bar{a}$	$\bar{b}$	$\overline{axb}$
0	0	0	1	1	1	1
1	0	1	0	0	1	0
0	1	1	0	1	0	0
1	1	1	0	0	0	0

We zien dat de 4e en 6e kolom gelijk zijn, waarmee het bewijs is geleverd.

We willen dit nu eens toepassen om in SERA toch een binaire optelopdracht te creëren, hoewel SERA normaal decimaal werkt.

Noemen we 2 overeenkomstige bits van 2 SERA woorden, die we willen optellen, x en y dan ontstaat allereerst een voorlopig sombit s en en een carry c, die als volgt samenhangen.

x	y	s	c
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

Met het voorgaande kunnen we s en c nu "logisch" uitdrukken in x en y, namelijk :

$s = x \times \bar{y} + \bar{x} \times y$

en $c = x \times y$

Met enkele EN en OF opdrachten kunnen we dus voor alle bits s en c bepalen. Verschuiven we daarna de c bits één plaats naar links en beschouwen we dan de s en c bits als nieuwe x en y bits, dan volgen hieruit weer nieuwe s en c bits. Na een aantal keren is de carry "uitgestorven" en vormen de s bits dan de definitieve binaire som.

In plaats van direkte formules voor $\bar{s}$ en c toe te passen werken we s eerst iets om en beschouwen speciaal $\bar{s}$.

$\bar{s}=\overline{x\cdot y+x\cdot y}=(\overline{x\ y})\ \ (\overline{x\ y})=$

$(\bar{x}+\bar{y})\ \ (\bar{x}+\bar{y})=(\bar{x}+y)\ \ (x+\bar{y})=$

$\bar{x}\ \ x+\bar{x}\ \bar{y}+x\ y+y\ \bar{y}=$

$\bar{x}\ \bar{y}=x\ y=\overline{x\ y}+x\ y$

Totaal gebruiken we nu :

$\bar{s}=\overline{x+y}+x\cdot y$

$c=x\ y$

Mocht men nog twijfelen aan de formule voor $\bar{s}$ dan kan men ook hier nog eenvoudig kontroleren door een waarheidstabel.

x	y	s	x+y	$\overline{x+y}$	x y	$\overline{x+y}+x\times y$	$\bar{s}$
0	0	0	0	1	0	1	1
1	0	1	1	0	0	0	0
0	1	1	1	0	0	0	0
1	1	0	1	0	1	1	1

Het volgende programma is uitgevoerd als subprogramma waarbij aangenomen dat bij aanroep de op te tellen getallen in A en B staan. Het resultaat wordt afgeleverd in A.

```
BINOPT   BPA    X          x ⟶ X
         OFA    ACCU-B     x+y ⟶ A
         ENB    X          C=x×y ⟶ B
         HNA    ACCU-A     x+y ⟶ A
         OFA    ACCU-B     s=x+y+x×y ⟶ A
         HNA    ACCU-A     S ⟶ A
         SOB    TERUG      klaar als alle c=0
         ------------
         BSA    X          s ⟶ X; nullen ⟶ A
         BSL    1          carry verschuiven
         OFB    ACCU-A     rondlopende carry verwerken
         HPA    X          opnieuw beginnen met s als x
         SAL    BINOPT+1   en c als y
X        0
         VRY    BINOPT
```

Merk op dat dit programma nu gebaseerd is op binair optellen in het 1 complement systeem. Door het weglaten van één instruktie wordt dit een optelprogramma in het 2 complement systeem (welke instruktie?)

Ir. D.H. Wolbers

Het is duidelijk dat men op deze wijze ook andere bewerkingen zoals binair aftrekken en desnoods vermenigvuldigen en delen kan simuleren. Om in het algemeen met alfanumerieke informatie als binair getal te kunnen rekenen komen binair optellen en binair aftrekken relatief nog zo vaak voor dat deze toch nog als 2 afzonderlijke opdrachten beschikbaar zijn.

BOP n binair optellen in B (B)+(n) B
BAF n binair aftrekken in B (B)-(n) B

Deze opdrachten werken in het *1 complementsysteem.*
Er bestaan geen equivalenten voor de A accumulator.

Juist bij gebruik van binaire opdrachten doet zich vaak de behoefte gevoelen om bepaalde patronen van nullen en enen direkt als constante via het invoerprogramma te kunnen invoeren.
Dit is mogelijk op de volgende manier. Men schrijft het SERA-woord als een zestientallig getal van max. 12 cijfers. Om deze schrijfwijze te kunnen aangeven wordt het gehele getal tussen ronde haakjes () geschreven. Voor de "cijfers" 10 t/m 15 worden gebruikt de letters A t/m F met

10 = A = 1010 13 = D = 1101
11 = B = 1011 14 = E = 1110
12 = C = 1100 15 = F = 1111

De volgende patronen

```
1111  1100  0010  1101  1111  0001  0000  1010  0101  1000
0000  0000  0000  0000  0000  0011  1111  1111  1111  1111
1111  0000  1111  0000  1111  0000  1111  0000  1111  0000
1111  1111  1111  1100  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000  0000  0000

0011  1011
1111  1111
1111  0000
0000  0000
0011  1111
```

kunnen we dan als volgt invoeren :

```
(FC2DF10A583B)
(3FFFFFF)
(F0F0F0F0F0F0)
(FFFC00000000)
(3F)
```

Niet significante nullen mogen dus worden weggelaten.

Het gebruik van de ronde haakjes hier is dus louter een conventie die voor het invoerprogramma voor de SERA is getroffen om dergelijke zestientallige getallen te kunnen onderscheiden van enerzijds normale opdrachten en anderzijds gewone decimale getallen.
Vergelijk bijvoorbeeld:

```
        BAF    3768  met  (BAF3768)
of            92376  met  (92376)
```

Met deze ronde haakjes conventie kan men ieder gewenst bitpatroon samenstellen en men zou dit dus ook kunnen gebruiken om in bepaalde gevallen in een woord alfanumerieke informatie vast te leggen. Dit laatste kan nodig zijn als men vaste stukjes tekst in het programma wil opnemen.

Dit is echter een omslachtige manier omdat men dan eerst van ieder alfanumeriek symbool de hexade code moet opzoeken, daarna het gehele woord opbouwen, dit weer in tetraden verdelen en dan zestientallig schrijven.

Neem als voorbeeld het woord moeilijk.

```
   M     O       E       I       L       I       J      K
010110  011000  001110  001010  010101  001010  010011 010100
```

Nu in tetraden verdelen.

```
0101  1001  1000  0011  1000  1010  0101  0100  1010  0100
1101  0100
```

Dit is dan te noteren als (59838A54A4D4)

Om deze complikatie te vermijden kent vrijwel ieder invoerprogramma daarom een conventie om direkt alfanumerieke informatie aan te geven.
In SERA is daarvoor gekozen dat de tekst tussen aanhalingstekens wordt geplaatst.

In het laatste voorbeeld dus :

```
      | "MOEILIJK"
```

In verband met de woordlengte van SERA kunnen dus tussen "maximaal 8 symbolen worden aangegeven. Let wel dat in dit geval niet zichtbare symbolen zoals spatie ook betekenis hebben.
Verder wordt aangenomen dat indien minder dan 8 symbolen gegeven worden links met nullen wordt aangevuld.

Als voorbeeld de tekst : dit is een voorbeeld.

```
"DIT IS E"
"EN VOORB"
"EELD    "
```

Let op dat in de laatste regel achter de D nog 4 spaties zijn overgelaten. Zouden we geschreven hebben

```
"EELD"
```

dan zou dat identiek geweest zijn met

```
"0000EELD"
```

Een eerste toepassing van voorgaande opdrachten bij invoerprogramma's is bijvoorbeeld het zoeken naar een bepaalde naam of criterium in een gegeven lijst.
Bij gebruik van symbolische adressen moet voor iedere naam worden bijgehouden welk echt adres daaraan is toegekend of nog moet worden toegekend.
Afhankelijk hoe het invoerprogramma is opgezet moeten meerdere lijsten worden bijgehouden bijv. een lijst van namen die zijn aangegeven met VRY en dus een blok definieren.

In SERA is voor namen een combinatie van maximaal 8 letters of cijfers toegelaten. Voor iedere naam moeten we dan in een tweede woord de gegevens opnemen.
Om ruimte te besparen kunnen we meerdere gegevens in één woord "samenpakken".
Moeten we in een dergelijke lijst zoeken naar gegevens volgens een bepaald criterium, dan heeft dat dus betrekking op gedeelten van woorden.
We spreken wel van het zoeken in een *TABEL* volgens een bepaald *MASKER*.

In verschillende machines bestaat een opdracht die deze bewerking direkt kan uitvoeren. Men dient dan op te geven het beginadres van de lijst, de lengte van de lijst, het masker en het te zoeken patroon. In SERA bestaat deze opdracht niet in direkte vorm, maar het volgende subprogramma verzorgt hetzelfde.

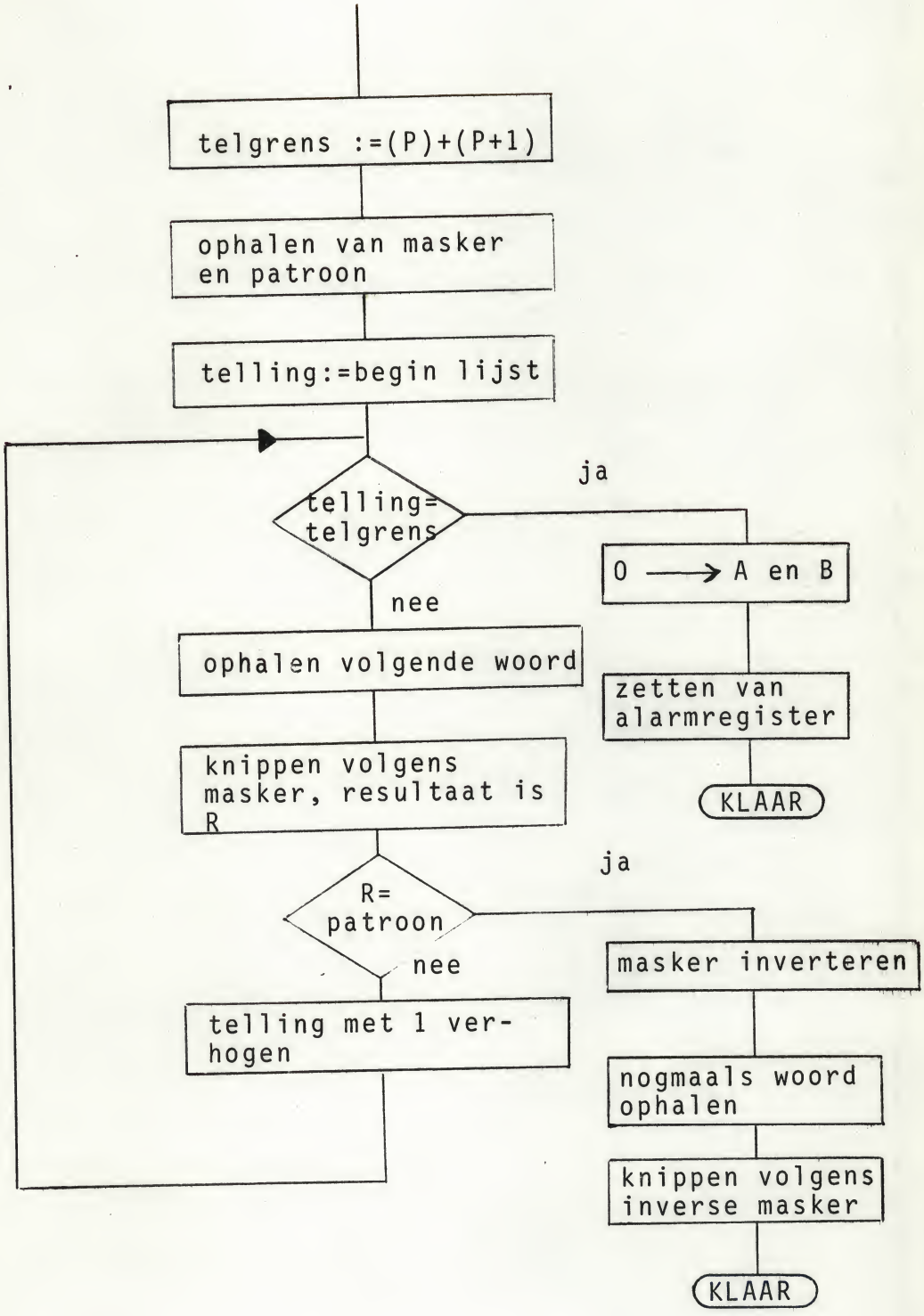
Bij aanroepen moet men in A meenemen een adres P, waarop op adres P en volgende adressen de volgende gegevens zijn vermeld.

P	begin adres van lijst
P + 1	lengte van de lijst
P + 2	masker
P + 3	patroon

Ir. D.H. Wolbers

Het masker is daarbij zo gegeven, dat dit enen bevat op de bitplaatsen die bij het zoeken gebruikt moeten worden en nullen op de overige bitplaatsen.
Het patroon bestaat uit enen en nullen, die samen de informatie vormen waarnaar gezocht moet worden. Natuurlijk moet het patroon passen binnen het masker.

Wanneer het programma een woord gevonden heeft dat aan de eisen voldoet, keert het terug met de informatie van de rest van het betrokken woord (dus waar het masker nullen heeft) in de B-accumulator. Bovendien wordt in A nog meegenomen het adres van het betrokken woord.
Komt het gezochte woord in de betrokken lijst niet voor, dan keert het subprogramma terug met nullen in A en B en bovendien wordt het alarmregister gezet. Het subprogramma kan worden opgeroepen onder de naam MASKZOEK. Aangezien dit soort werkzaamheden ook voor een elektronische rekenmachine tamelijk tijdrovend is, heeft het zin de werkelijke zoekcyclus zo snel mogelijk te laten aflopen, dus met zo min mogelijk opdrachten. Het programma begint daarom met het ophalen van de nodige gegevens, zodat die binnen de zoekcyclus niet iedere keer met modificatie-opdrachten gehaald moeten worden.
(zie volgende pagina)


telgrens :=(P)+(P+1)
ophalen van masker en patroon
telling:=begin lijst
telling= telgrens
ja
nee
0 → A en B
zetten van alarmregister
KLAAR
ophalen volgende woord
knippen volgens masker, resultaat is R
R= patroon
ja
nee
masker inverteren
telling met 1 verhogen
nogmaals woord ophalen
knippen volgens inverse masker
KLAAR

Ir. D.H. Wolbers

```
MASKZOEK  MOD   ACCU-A   ]
          HPB   0        }
          MOD   ACCU-A   }   eerste adres voorbij lijst
          OPB   1        }   naar telgrens
          BPB   TELGR    ]
          MOD   ACCU-A
          HPB   2
          BPB   MASKER       masker naar MASKER
          MOD   ACCU-A
          HPB   3
          BPB   PATR         patroon naar PATR
          MOD   ACCU-A
          HPA   0
CYCLUS    HPB   TELGR
          SAB   EINDETEL     spring als telgrens is bereikt
          MDD   ACCU-A
          HPB   0            ophalen volgende woord
          ENB   MASKER       knippen volgens masker
          BAF   PATR     }
          SOB   GEVONDEN }   vergelijken met patroon
          OPA * 1            telling met 1 verhogen
          SAL   CYCLUS
EINDETEL  HPA * 0            nullen naar A
          HPB * 0            nullen naar B
          DLN * 0            zet alarmregister
          SAL * TERUG
GEVONDEN  HPB   MASKER   }
          BNB   MASKER   }   masker inverteren
          MOD   ACCU-A
          HPB   0
          ENB   MASKER       knippen met inverse masker
          SAL   TERUG
TELGR     0
MASKER    0
PATR      0
          VRIJ  MASKZOEK
```

Terwijl het invoerprogramma dus van veel belang kan zijn voor het vereenvoudigen van programmeren is dit toch niet het enige hulpmiddel

ENKELE ANDERE TECHNIEKEN

We geven daarom ook in het volgende ook de principes van enkele andere technieken.
Allereerst de *INTERPRETEERTECHNIEK*. Deze techniek is vooral in het verleden meer toegepast voor wetenschappelijke berekeningen dan voor administratieve toepassingen. Zij is ontstaan op grond van de volgende overwegingen. Wanneer in een programma zeer vaak dezelfde bewerkingen moeten worden uitgevoerd bijv. berekeningen met drijvende komma of in dubbele lengte, dan is dit in principe uitvoerbaar

met behulp van dergelijke bewerkingen niet beschikbaar subprogrammatechnieken voor zover opdrachten voor direkte uitvoering van dergelijke bewerkingen niet beschikbaar zijn. Dit was vooral met de oudere machines het geval. Denken we bijv. aan subprogramma's voor optellen,aftrekken, vermenigvuldigen en delen van dubbele lengte getallen. Bij het oproepen van dergelijke programma's moeten 4 gegevens worden meegenomen, terwijl het resultaat als 2 woorden wordt opgeleverd. Dit betekent dat in de regel iedere subprogramma-oproep gepaard gaat met ongeveer 10 transportopdrachten. Dit aantal kunnen we aanzienlijk beperken door uitbreiding van de subprogramma's in die zin dat de gewenste transporten ook door de subprogramma's worden verzorgd. Dit komt er op neer dat men in een soort gecodeerde vorm aangeeft welke bewerkingen in serie verricht moeten worden. Dit kan bijv. in de vorm van een reeks codegetallen. Het betrokken subprogramma moet dan deze codegetallen achtereenvolgens ophalen, analyseren en afhankelijk daarvan bepaalde bewerkingen uitvoeren. De genoemde codegetallen kunnen dus ook als een soort instrukties worden opgevat of wel "geïnterpreteerd".
Een dergelijk programma, dat deze mogelijkheid dan ook geeft noemt men wel *interpreteerprogramma*.
In principe is de keuze van codegetallen vrij, het interpreteerprogramma moet slechts aangepast zijn aan deze keuze. Dit betekent, dat men programma's geschreven in een andere code dan voor de betrokken geldig is toch kan verwerken, hoe afwijkend de code van de gegeven machinecode ook is.
Daarmee openen zich twee toepassingsgebieden voor deze interpreteertechniek :

1. *Men kiest als nieuwe code een code, die de gebruiker meer aanspreekt en gemakkelijker is te hanteren.*
2. *Men kiest als nieuwe code zonder meer de code van een andere machine.*

Voor beide toepassingen dient men daarbij het volgende in het oog te houden. Zowel het te interpreteren programma als het interpreteerprogramma zelf moeten gelijktijdig in het geheugen van de machine aanwezig zijn. De verwerking geschiedt nu relatief langzamer dan wanneer voor hetzelfde probleem een programma in de machinecode zelf zou zijn samengesteld. Dit spreekt vooral aan wanneer men lussen in het te interpreteren beschouwt. Immers elke opdracht, die in de lus voorkomt moet bij iedere doorgang steeds opnieuw volledig geanalyseerd (geïnterpreteerd) worden. Vooral om deze reden is het eerste genoemde toepassingsgebied namelijk een makkelijker code wel beperkt. Dit komt omdat men in dit geval een alternatief heeft in de vorm van de vertaaltech-

niek, die nog hierna ter sprake komt.

Het tweede toepassingsgebied n.l. de interpretatie van een andere machinecode is vooral van belang bij de overschakeling naar een andere machine.
Heeft men in een gedeeltelijk geautomatiseerd bedrijf een computer, waarop reeds een aantal werkzaamheden regelmatig plaatsvinden, dan doet zich een moeilijkheid voor wanneer men de bestaande rekenmachine wil gaan vervangen door een modernere, in het algemeen wel grotere en snellere machine. Tot op heden betekende dit vrijwel altijd de omschakeling op een geheel andere code. Dit impliceert dat men alle bestaande programma's, waarin vaak vele manjaren werk gestoken zijn, moet omcoderen en dat dan bovendien nog in een veelal zeer korte overgangsfase.

Heeft men voor de nieuwe machine de beschikking over een interpreteerprogramma, die de code van de oude machine kan interpreteren, dan is men in staat toch de bestaande programma's ook in de nieuwe machine zonder wijziging te gebruiken. Men simuleert als het ware de oude machine op de nieuwe. Omdat de oude machine wat basissnelheid betreft meestal langzamer is dan de nieuwe valt bovendien het relatief langzaam werken niet zo direkt op. Wel moet men zich natuurlijk realiseren dat voor programma's die regelmatig gebruikt worden en voorlopig ook in gebruik blijven, het wel lonend is om deze om te coderen in de nieuwe machine omdat men dan pas het volle profijt van de nieuwe machine trekt.
Op deze wijze kan men echter wel de daarmee gepaard gaande programmatische arbeid over wat langere termijn uitsmeren en bovendien dan combineren met veranderingen in de programma's, die men uit gebruiksoverwegingen toch wel had willen aanbrengen.

DE PLUIZER

Een bijzonder geval van het interpreteren van een andere machinecode is nog dat men als te interpreteren code kiest de code van de machine zelf. Aan het interpreteerprogramma voegt men dan echter nog iets toe n.l. na de interpretatie van iedere opdracht drukt men af welke opdracht zojuist verwerkt is en welke invloed dit eventueel gehad heeft op bepaalde registers. In SERA bijv. de veranderingen in A en B accumulator. Met deze toevoeging is een dergelijk programma dan een machtig hulpmiddel bij het testen van een nieuwe programma. Men krijgt dan gedrukt alle details hoe het programma dynamisch door de machine verwerkt zou zijn, ware dit rechtstreeks door de machine uitgevoerd.

Ir. D.H. Wolbers

Dergelijke programma's staan meestal bekend onder de naam *tracer* of *pluizer*.

In het volgende voorbeeld is het principe van een PLUIZER voor SERA-code gegeven.
Behalve het te testen programma wordt ook het programma PLUIZER in het geheugen gebracht. Dit laatste programma wordt nu gestart. Het begint met een kaart te lezen. Hierop staan 2 getallen geponst. Het eerste geeft een adres in het nieuwe programma aan waar het interpreteren moet beginnen. Het tweede is het normale startadres van het nieuwe programma. Uit het nieuwe programma wordt op de aangegeven plaats een instruktie weggehaald en vervangen door een spronginstruktie naar de label BEGIN in het PLUIZERprogramma. Daarna wordt een sprong uitgevoerd naar het begin van het nieuwe programma.

Dit programma begint op de normale manier te werken alsof het gestart was met SRT.
Dit gaat door tot het punt bereikt wordt waar een instruktie is weggenomen en vervangen door een spronginstruktie naar label BEGIN van het PLUIZERprogramma. Vanaf dit moment begint het eigenlijke werk van het PLUIZERprogramma.
Deze brengt allereerst de inhoud van alle echte registers over naar overeenkomstige schijnregisters, die op zich echter normale geheugenplaatsen zijn. Dan wordt de weggenomen instruktie hersteld en daarna begint het interpreteren. Te beginnen met de oorspronkelijke weggenomen instruktie wordt iedere instruktie opgehaald, ontleed en de gewenste bewerkingen uitgevoerd in de schijnregisters. Bovendien vindt iedere keer een subprogrammasprong naar een drukprogramma plaats, dat verondersteld wordt de gewenste uitvoer te geven, maar hier verder niet is vermeld.
Om misverstand te voorkomen wordt nog opgemerkt dat het volgende PLUIZERprogramma verre van volledig is en met name de invoed van de kenmerk-tetrade buiten beschouwing laat. Het volledige programma is echter alleen veel uitgebreider maar bevat geen principiëel andere delen.

```
PLUIZER  LSK
         HAB    1:4
         KAG    4
         BPB    WERK      }
         MOD    WERK      }   neem eerste te interpreteren
         HPA    0         }   instruktie weg en plaats daar
         BPA    WERK+1    }   een sprong naar BEGIN
         HPA    HULP      }
         MOD    WERK      }
         BPA    0         }
         HAB    11:14     }
         KAG    4         }   spring naar het te onderzoeken
         SAL    ACCU-B    }   programma
```

Ir. D.H. Wolbers

```
BEGIN   BPA   A
        BPB   B
        HPA * 1
        SIA   PT1        breng de gegevens van de echte
        HNA * 1          register over naar de schijn-
PTI     BPA   ALARM      registers
        HPA   TERUG
        BPA   T
        HPA   WERK+1     herstel de weggenomen instruktie
        MOD   WERK
        BPA   0
        HPA   WERK       opdrachtenteller wordt voor de
        BPA   OPT        1e keer gevuld
PT2     SSP   DRUK       druk alle gewenste informatie
        MOD   OPT        haal op de volgende te inter-
        HPA   0          preteren instruktie
        BPA   INSTRUC
        HPB   ACCU-A
        ENB   HULP+1     in B nu alleen de operatie
        AFA   ACCU-B     in A nu alleen het adres
        BSA   ADRES
        ABL   7          operatie nu rechts in A
        MOD   ACCU-A     uitsplitsen volgens
        SAL   TABEL      type opdracht
WERK    0
        0
HULP    SAL   BEGIN
        (FF0000)
A       0
B       0
T       0
ALARM   0
OPT     0
INSTRUC 0
ADRES   0
PT3     HPA   OPT        opdrachtenteller met 1
        OPA * 1          verhogen
        BPA   OPT
        SAL   PT2
TABEL   SAL   SAL        voor opdracht 00
        SAL   SPA        voor opdracht 01
          :
        SAL   SSP        voor opdracht 07
          :
        SAL   HPB        voor opdracht 22
          :
        SAL   OPA        voor opdracht 30
          :
        enz.
SAL     HPA   ADRES
```

```
        BPA   OPT
        SAL   PT2
SPA     HPA   A
        SPA   SAL
        SAL   PT3
HPB     MOD   ADRES
        HPB   0
        BPB   B
        SAL   PT3
OPA     HPA   A
        MOD   ADRES
        OPA   0
        BPA   A
        HPA * 1
        SIA   PT4
        HNA * 1
PT4     BPA   ALARM
        SAL   PT3
SSP     HPA   OPT
        OPA * 1
        BPA   T
        SAL   SAL
```

Volledigheidshalve kan worden vermeld dat juist door deze interpretatietechniek het mogelijk is ook een machine als SERA die technisch niet bestaat toch te simuleren op bestaande machines, waardoor het opeens toch mogelijk wordt SERA programma's te verwerken. Dit is dan natuurlijk geschikt voor oefenprogramma's.

Wat betreft de toepassing van interpreteren om een gemakkelijker code te hanteren is reeds opgemerkt dat men hier in vele de voorkeur geeft aan een andere programmeertechniek namelijk het vertalen.
Het principe kan men als volgt omschrijven. Voor een probleem wordt een programma geschreven in een betrekkelijk eenvoudig te leren programmeercode, in dergelijke gevallen ook veelal aangeduid als programmeertaal. De taal noemt men dan de *brontaal* en het hierin geschreven programma het *bronprogramma*.

Nu zijn de definities en regels voor een dergelijke taal wel zo getroffen dat iedere uitdrukking in zo'n taal een eenduidige betekenis heeft, d.w.z. niet voor tweeërlei uitleg vatbaar is. Daardoor is het mogelijk een dergelijk *bronprogramma* om te zetten in een equivalent programma direkt in *machinecode*.
Deze code noemt men dan wel *objectcode* of objecttaal en het hierin samengestelde programma dan het *objectprogramma*.

Het omzetten van de ene code in de andere, dat weliswaar in bepaalde gevallen bijzonder gecompliceerd kan zijn is

overigens toch een automatisch proces dat op zich door een computer kan worden uitgevoerd.

Het daarvoor benodigde programma noemt men dan *vertaler*, veelal ook translator of *compiler*. Dit laatste drukt als het ware uit dat het programma opnieuw wordt "samengesteld". Het grote voordeel van dit systeem is dat de analyse van het gegeven programma slechts éénmaal plaatsvindt, ook de opdrachten,die in een lus voorkomen. De som van de vertaaltijd en de verwerkingstijd van het objectprogramma is daarom in de meeste gevallen veel kleiner dan in het geval het gegeven bronprogramma rechtstreeks door een interpreteerprogramma verwerkt zou zijn.
Men spreekt ook wel over het gebruik van een zogenaamde *autocode*. Dit is dus een programmeertaal, die enerzijds zo nauw mogelijk aansluit bij onze gebruikelijke menselijke schrijfwijze en anderzijds zo exact en precies is, dat het door een machine middels een speciaal vertaalprogramma begrepen kan worden.

Van deze autocodes bestaan verschillende uitvoeringen.Het bestaan van meerdere van dergelijke universele programmeertalen is voor een deel een kwestie van geschiedenis, omdat ze vanuit en door verschillende firma's het met de huidige stand der techniek van de machines en de kennis van programmeren niet, of althans nog niet mogelijk is één werkelijk universele programmeertaal te gebruiken.

Het gebruik van zo'n taal zou ten eerste vereisen, dat de meest uiteenlopende problemen in deze taal geformuleerd kunnen worden en ten tweede voor iedere machine een vertaalprogramma vergen, dat zo goed is, dat steeds de meest efficiënte machinecodeprogramma's ontstaan.
Het laatste is voorlopig een wensdroom en in de praktijk zijn daardoor een aantal programmeertalen ontstaan, die ieder voor zich meer zijn gericht op een bepaald toepassingsgebied.

Dergelijke talen zijn ook aan bepaalde beperkingen onderhevig en dit alles maakt het dan mogelijk redelijke effectieve vertalers te bouwen. Door het meer gerichte gebruik spreekt men wel van probleem georiënteerde programmeertalen.
Voorbeelden voor wetenschappelijk gebruik zijn FORTRAN (formula translation) ontwikkeld door IBM en een meer internationaal overeengekomen taal ALGOL 60 (algorithmic language).
De toevoeging 60 duidt op het jaartal, waarin de conventies zijn vastgesteld, dit ter onderscheiding met toekomstige uitvoeringen. FORTRAN heeft vooral veel ingang gevonden in Amerika en daar niet alleen voor IBM machines,

maar ook voor andere merken computers. In Europa is het naast FORTRAN vooral ALGOL dat toepassing vindt.

Voor administratieve toepassingen bestaan enkele minder gebruikte talen van verschillende fabrikanten en vindt tegenwoordig vooral opgang de algemeen overeengekomen taal COBOL (common business oriented language). Het grote verschil tussen ALGOL 60 en COBOL is juist gelegen in de veel uitvoeriger mogelijkheden van omschrijving en gebruik van input-output apparaten bij de laatste taal. Een speciale eis bij COBOL is nog geweest, dat een in COBOL geschreven programma ook nog redelijk leesbaar en begrijpelijk moet zijn voor mensen, die wel bekend zijn met het probleem maar niet met programmeren.

Bij de hiervoor gegeven voorbeelden is het gebruik van woorden als programmeertaal, vertalen, e.d. niet aan twijfel onderhevig. Omgekeerd wanneer men een programma schrijft in direkte machinecode, dan zal men spreken over invoerprogramma of eventueel assembler. Zelfs bij gebruik van hulpmiddelen als mnemotechnische codering en symbolische adressen blijft men toch in het algemeen spreken over invoerprogramma. Gaat men echter nog verder en gaat men bij name generators in het invoerprogramma inbouwen dan begint langzamerhand het onderscheid tussen een gecompliceerd assembler en een eenvoudige vertaler te vervagen.
Wat is nu een generator? Het woord drukt eigenlijk al de werking uit namelijk een generator of beter gezegd een programma-generator genereert een stuk programma, overigens volgens voorgeschreven richtlijnen.

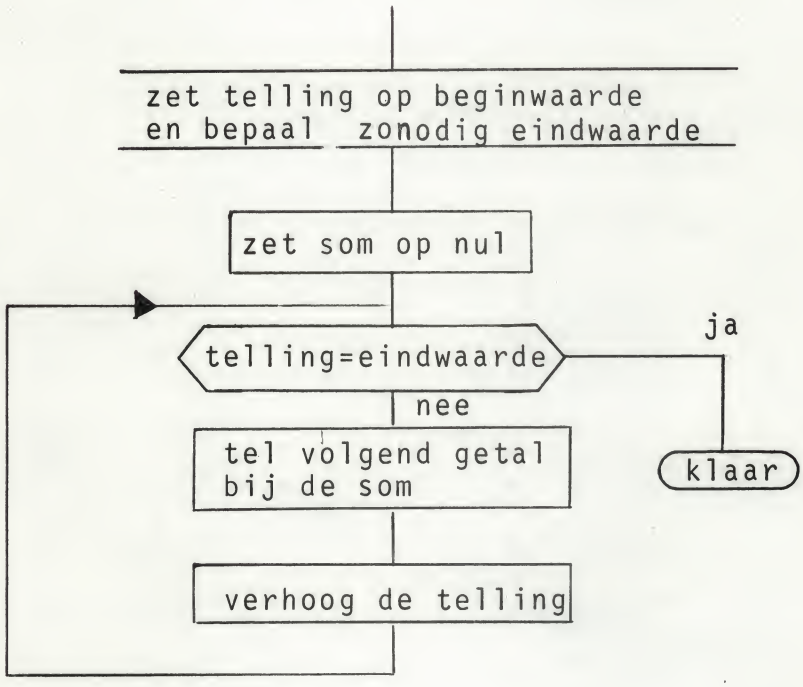
Wij zullen dit aan de hand van enkele stukjes SERA programma verduidelijken.
In een totaal-programma komt het vaak voor dat een bepaalde bewerking op verschillende plaatsen in het programma nodig is, zij het dan steeds met andere getallen of in het algemeen gesproken voor verschillende data. In dat geval denken we dan direkt aan de reeds eerder besproken subprogramma's.
We kiezen als voorbeeld het bepalen van de som van een aantal getallen, die op achtereenvolgende adressen in het geheugen staan en waarvan het aantal bekend is.
Neem aan de getallen staan in het geheugen vanaf het symbolische adres LYST, het aantal getallen staat aangegeven op de geheugenplaats met symbolisch adres AANTAL en de som van de getallen moet komen op de geheugenplaats met de naam SOM.
De gewenste bewerking kan dan verkregen worden door de volgende instruktie.

Ir. D.H. Wolbers

```
HPA  *  LYST
HPB     AANTAL
SSP     OPTELLEN
BPB     SOM
```

Daarbij is aangegeven dat er een subprogramma bestaat met de naam OPTELLEN met de volgende conventies. Bij aanroepen staat in ACCU-A het adres van het eerste getal van de serie waarvan de som moet worden bepaald. In ACCU-B staat het aantal getallen. Verder levert het subprogramma na terugkeer in ACCU-B de gevraagde som.

We zullen in dat geval ook het subprogramma beschouwen en geven daartoe eerst het stroomdiagram.



Volgens dit schema kan het subprogramma er als volgt uitzien.

```
OPTELLEN | OPB   ACCU-A
         | BSB   TELGRENS
         | BPB   TOTAAL
OPN      | HPB   TELGRENS
         | SAB   KLAAR
         | HPB   TOTAAL
         | MOD   ACCU-A
         | OPB   /0
```

Ir. D.H. Wolbers

```
                BPB    TOTAAL
                OPA *  1
                SAL    OPN
KLAAR           HPB    TOTAAL
                SAL    TERUG
TELGRENS        0
TOTAAL          0
                VRY    OPTELLEN
```

Het is goed ons te realiseren dat in dit geval het subprogramma 15 geheugenplaatsen in beslag neemt en dat voor iedere oproep in feite 4 instrukties nodig zijn. Daar komt bij dat we door de gekozen vorm van subprogrammatechniek ook gedwongen worden tot bepaalde programmatische stappen, die niet zozeer afhangen van het probleem als wel van de gekozen programmeringsconventies.
Vergelijk daartoe de eerste opdracht OPB ACCU-A.

Daarmee brengen we in rekening dat we de telling zullen laten lopen vanaf het eerste adres van de lijst tot en met het laatste.
Op zichzelf zou het meer voor de hand gelegen hebben de telling te laten lopen van 0 tot het gegeven aantal. Dat geeft in dit geval echter moeilijkheden bij de verderop gebruikte modificatie-opdracht, waarmee we steeds een volgend exemplaar optellen.

Nu zou men kunnen opmerken, dat dit toch een gevolg is van de mogelijkheden of in dit geval misschien beter gesteld beperkingen van de SERA code. Bij een andere machine zou men dit geval dan misschien wel eleganter kunnen oplossen. Bij een dergelijke machine zullen echter ongetwijfeld weer andere situaties kunnen optreden, die juist voor die machine weer ongunstiger uitvallen.

Hiermee wil slechts gezegd zijn, dat het gebruik van een subprogramma niet in alle opzichten de voordeligste oplossing is.

Laten we dit voorbeeld daarom ook programmeren zonder gebruik van een subprogramma.
Dit betekent dat in plaats van de 4 instrukties nodig voor de oproep nu direkt de benodigde instrukties voor de bewerking zelf in de plaats komen.
Dit stukje programma zou er als volgt uit kunnen zien.

```
OPA             HPA *  0
                BPA    SOM
                HPB    AANTAL
                SAB    VERDER
                HPB    SOM
                MOD    ACCU-A
```

Ir. D.H. Wolbers

```
          OPB   LYST
          BPB   SOM
          OPA * 1
          SAL   OPN
VERDER    VRY   OPT
```

Het geheel kost nu 10 instrukties. Het nadeel is natuurlijk dat we op iedere plaats in het programma waar we deze bewerking willen uitvoeren een dergelijke serie instrukties nodig hebben en die ook zelf volledig moeten uitschrijven. Willen we echter toch deze werkwijze volgen dan kunnen we het laatst genoemde nadeel echter opvangen door het toepassen van een generator. Daartoe moeten we eerst in ons programma de steeds te genereren serie instrukties definiëren en daarna op iedere plaats waar we die willen hebben op speciale manier daartoe de wens te kennen geven. Men spreekt in dat geval wel over het gebruik van *macro-opdrachten*, of kortweg macro's. Men onderscheidt dan verder de macrodefinitie, waar dus een dergelijk stuk programma gedefiniëerd wordt en de macro-oproep, die aangeeft dat daar ter plaatse een dergelijk stuk programma geproduceerd of anders gezegd gegenereerd moet worden. Het deel van de assembler, dat voor deze bewerking zorgt noemt men dan wel de generator.

Wanneer we zouden aannemen dat ook de SERA assembler met dergelijke mogelijkheden was uitgerust dan zouden we het voorgaande voorbeeld als volgt kunnen programmeren.

Het definiëren van de macro gebeurt als volgt :

```
          MDF   OPTELLEN,L,N,S;
          HPA * 0
          BPA   S
OPN       HPB   N
          SAB   VERDER
          HPB   S
          MOD   ACCU-A
          OPB   L
          BPB   S
          OPA * 1
VERDER    SAL   OPN
VERDER    MEN   OPTELLEN
```

Hierin ontmoeten we 2 nieuwe mnemotechnische codes n.l. MDF en MEN. Deze dienen om het begin en het einde van een programmaskelet te markeren. Daarbij staat MDF voor macro definitie en MEN voor macro einde.
Als verdere conventie is aangenomen dat achter MDF volgt een zelf gekozen symbolische naam, die dan tevens de naam van de betrokken macro inhoudt en waaronder deze macrode-

finitie later kan worden aangeroepen.
Achter de macronaam volgen gescheiden door komma's een aantal zogenaamde parameters. Dit zijn allemaal namen die gebruikt worden om het daarop volgende stukje programma te kunnen beschrijven, maar waarvoor de werkelijke namen of betekenissen pas bepaald worden in de actuele situatie waar de macro nodig is.
In dit voorbeeld is de macronaam dus OPTELLEN en de 3 parameters zijn L, N en S. De regels tussen MDF en MEN zien er uit als een normaal stukje SERA programma.
Men moet echter bedenken dat wanneer een nieuw programma wordt ingelezen in het algemeen door de eerste BGN opdracht bepaald wordt vanaf welke plaats in het geheugen opgeborgen zal worden. Komt er nu in de gehele serie aangeboden instrukties plotseling een MDF, dan is dit een aanwijzing voor het invoerprogramma. Vanaf dat moment wordt iedere instruktie in ongewijzigde of vrijwel ongewijzigde vorm opgeborgen ergens in de werkruimte van het invoerprogramma.
Dit gaat door tot de aanduiding MEN en de hierna volgende regels worden weer gewoon sequentieel vanaf het vorige onderbrekingspunt in het geheugen opgeborgen.
In het programma zelf is dus nog niets van deze bewerking opgenomen. Er kunnen zelfs nog wel meer macrodefinities gegeven worden en die worden allemaal binnen de ruimte van het invoerprogramma opgeslagen.
Daarmee wordt ook de macronaam duidelijk omdat iedere macro natuurlijk eenduidig bepaald moet kunnen worden. Evenals bij symbolische adressen zijn dus gelijke namen verboden.

We beschouwen thans de positie in het programma waar we een dergelijke serie instrukties tussengevoegd willen hebben.
Dit kan aangegeven worden door

```
OPT  |   OPTELLEN, LYST, AANTAL, SOM ;
```

In het programma schrijven we slechts een regel.Het effekt is echter dat door het in werking treden van de programgenerator als onderdeel van het invoerprogramma op deze plaats in ons programma 10 instrukties worden tussengevoegd zoals het eerder gegeven skelet aangeeft.
Dit gebeurt dan met dien verstande, dat overal waar in het skelet de naam L gebruikt werd deze nu vervangen wordt door LYST.
Hetzelfde geldt voor N door AANTAL en S door SOM. Wanneer we deze transformatie uitvoeren dan zien we dat precies het stukje programma gegenereerd wordt dat we eerder ook reeds hadden aangegeven. Het voordeel van deze werkwijze is echter dat we nu overal in het programma waar we een dergelijk stuk programma wensen, kunnen volstaan met één

regel.
Bijvoorbeeld, we willen een stukje programma om de som te bepalen van alle getallen in het geheugen vanaf adres TABEL, het aantal wordt gegeven op symbolisch adres LENGTE en de gevraagde som moet geplaatst worden op symbolisch adres RESULTAAT.

Dit kunnen we dan creëren door

```
BERSOM  | OPTELLEN, TABEL, LENGTE, RESULTAAT ;
```

Deze regel zal dan door de generator vervangen worden door

```
BERSOM  | HPA * 0
        | BPA   RESULTAAT
OPN     | HPB   LENGTE
        | SAB   VERDER
        | HPB   RESULTAAT
        | MOD   ACCU-A
        | OPB   TABEL
        | BPB   RESULTAAT
        | OPA * 1
        | SAL   OPN
VERDER  | VRY   BERSOM
```

OPEN-EN GESLOTEN-SUBPROGRAMMA'S

Voor de programmeur heeft het gebruik van macro's dus wel een sterke analogie met het gebruik van subroutines. Dit gaat zelfs zover dat men vaak ook in deze gevallen wel spreekt van een subprogramma.
Om het onderscheid echter duidelijk te maken noemt men deze vorm van subprogramma's dan echter "open" subprogramma's en de vroeger beschreven vorm "gesloten" subprogramma's.
Men moet dus goed onderscheiden :

1. Gesloten subprogrmma's vormen een deel van het totale programma. Tijdens de uitvoeringsfase van het programma worden ze dynamisch opgeroepen vrijwel altijd met een speciaal soort absolute sprong, de subprogrammasprong, in SERA met de code SSP.
2. Een open subprogramma is een stuk programma dat tevoren gedefiniëerd is en een naam heeft gekregen en tijdens het inlezen van het totale programma op iedere plaats in het programma wordt tussengevoegd waar dat expliciet is aangegeven.

Na assemblage van het totale programma is een open subprogramma dan ook niet meer als zodanig te herkennen. Met de open subroutine zijn tevens verbonden de namen macro-

I. D.H. Wolbers

definitie en macro-oproep.

Moderne assemblers die het hiervoor beschreven systeem van macrodefinities toelaten maken meestal nog onderscheid tussen enerzijds gewone, of eigen of gedefiniëerde macro's en anderzijds systeemmacro's.
De eerste soort is dan zoals hiervoor aangegeven.
Bij de tweede soort, de systeemmacro's, kent men geen macrodefinities.
De definitie is dan bij voorbaat al in de assembler ingebouwd.
Dergelijke macro's kan men dan zonder meer in eigen geschreven programma's gebruiken. Men past deze vorm vooral toe voor stukjes standaard-programma, die betrekking hebben op in- en uitvoer. Dit laatste ruim genomen zodat daar ook onder valt gebruik van magneetbanden, schijven enz.

Het gebruik van macro's zowel zelfgedefiniëerde als systeemmacro's vindt steeds meer ingang. De mogelijkheden zijn ook bijzonder groot. Vooral wanneer men in de mogelijkheden ook verder gaat.
Zo kan men bijvoorbeeld gaan toelaten, dat in het skelet, ook vaak aangegeven met body, van de macro op zich weer andere macro-oproepen mogelijk zijn.
Verder is het mogelijk voorwaardelijke macro-opdrachten in te voeren, waardoor de wijze waarop later een programma gegenereerd zal worden nog afhankelijk zal zijn van bepaalde parameters.
Zo zijn nog wel meer mogelijkheden denkbaar die echter buiten het bestek van deze kursus vallen. Wel kan hier nog worden opgemerkt dat zoals overal bij gebruik van rekenmachines men voor meerdere faciliteiten ook een zekere prijs betaalt.
In dit geval wordt natuurlijk de assembler steeds ingewikkelder. Dat betekent een groter stuk geheugen dat nodig is voor de assembler en verder moet er meer verricht worden tijdens het inlezen, dus zal dit inlezen ook meer tijd vergen.

Tenslotte is het zo zoals reeds eerder opgemerkt; een assembler met vergaande macrofaciliteiten is praktisch een vertaler of compiler geworden. Men kan tenslotte een compiler ook opvatten als een invoerprogramma met een groot aantal ingebouwde systeemmacro's, waarbij het in te voeren programma nog slechts bestaat uit een aaneenschakeling van macro-oproepen. Wanneer men de namen van deze macro's dan bovendien slim kiest dan wordt deze aaneenschakeling bijna een lopend verhaal.

Stel bijv. dat men als conventie aanneemt dat spaties tussen woorden dezelfde betekenis hebben als scheidings-

komma's. Het is dan zeker mogelijk de zin

```
ADD      A    TO    B
```

op te vatten als een macro-oproep en daaruit gegenereerd of anders gezegd vertaald te krijgen in SERA termen

```
HPA      B
OPA      A
BPA      B
```

Volgens bovenstaande gedachtengang zou er dus een geleidelijke overgang bestaan van normale assembler tot compiler. In de praktijk vat men dat meestal niet zo op en spreekt men liever van een assembler met macrofaciliteit wanneer principiëel toch eigenlijk nog geprogrammeerd wordt in de machinecode.
Men spreekt van compiler als het totale bronprogramma geschreven wordt in een code of taal, die niets meer uitstaande heeft met de machinecode en in principe zelfs onafhankelijk is van de betrokken machinecode.

KANAALORGANISATIE, MULTIPROGRAMMING, MULTIVERWERKING

Prof. Ir. D.H. Wolbers

Een publikatie van de Stichting Het Nederlands Studiecentrum voor Informatica, Amsterdam.

Ir. D.H. Wolbers

KANALEN

Bij het bespreken van de magneetbanden is reeds aangegeven dat deze via speciale apparaten, kanalen genaamd, aan het geheugen van de machine worden aangesloten.
We willen hier deze kanalen en de daarmee samenhangende kanaalorganisatie wat nauwkeuriger bekijken.

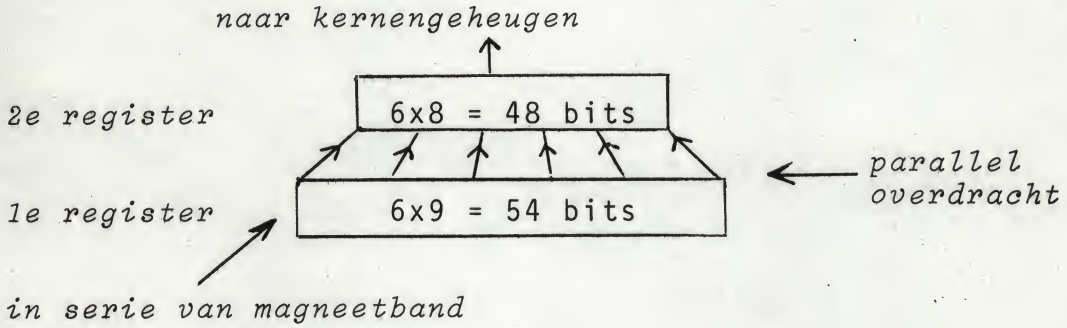
Hoewel daarbij speciaal het voorbeeld van informatietransport tussen kerngeheugen en magneetbanden ter sprake komt, geldt voor moderne rekenmachines dat ook het transport tussen kerngeheugen en andere in- en uitvoerapparaten zoals kaartlezers, regeldrukkers, schijvengeheugens e.d. op gelijke wijze behandeld worden.
De reden hiervoor ligt in de eerste plaats in het snelheidsverschil tussen de in- en uitvoerapparaten enerzijds en het geheugen, besturingsorgaan en rekenorgaan anderzijds.

Laten we als voorbeeld nemen het overbrengen van een blok gegevens van 500 woorden van magneetband naar geheugen. Nemen we aan dat het hier een woordlengte betreft van 48 bits zoals in SERA en verder dat de magneetband is ingericht met 9 sporen, waarvan 8 informatiesporen terwijl de negende gebruikt wordt voor pariteitskontrole.
Als lees- en schrijfsnelheid voor de magneetband nemen we 60000 karakters per sekonde. Nemen we verder aan dat een eventuele langpariteitskontrole alleen per blok wordt uitgevoerd, dan is deze wat de snelheidsbeschouwingen betreft te verwaarlozen.
Op de magneetbanden moeten we dan rekenen met 6 karakters per geheugenwoord van 48 bits. Een blok van 500 woorden betekent 3000 karakters op de band. De transporttijd hiervoor bedraagt 3000/60000sek.=1/20 sek.= 50000 mikrosek. Nemen we als geheugencyclustijd 2 mikrosek. dan is het kernengeheugen gedurende 500x2=1000 mikrosek. bezet voor het werkelijk inlezen. Gedurende de gehele transporttijd is het geheugen zelf dus slechts 1/50 deel van de tijd nodig. Bij een gemiddelde instruktieduur van 10 mikrosek, 2 mikrosek voor ophalen van de instruktie, 2 mikrosek voor ophalen operand, totaal 14 mikrosek, zouden dan in de genoemde wachttijd nog 49000/14 = 3500 instrukties kunnen worden uitgevoerd. Om dit te realiseren is echter een speciale organisatie nodig waarbij de kanalen een speciale rol spelen.
Immers bij de afwikkeling van niet in- en uitvoeropdrachten werkt het besturingsorgaan in 3 fasen. Ten eerste ophalen van een nieuwe instruktie, daarna zorgen voor de uitvoering hiervan en tenslotte ophogen van de opdrachtenteller. Voor omvangrijke opdrachten overweegt daarbij vooral de 2e fase wat tijdsduur betreft.

Ir. D.H. Wolbers

Zouden we nu voor in- en uitvoeropdrachten ook zonder meer deze werkwijze volgen dan stond het besturingsorgaan het grootste deel van de tijd te wachten en zou er van een parallelwerking geen sprake kunnen zijn. Bij een in- of uitvoeropdracht wordt daarom wat het besturingsorgaan betreft de tweede fase beperkt tot het melden van de gewenste in- of uitvoer aan het daarbij betrokken kanaal. Zodra het kanaal alle benodigde gegevens heeft ontvangen meldt het dit terug aan het besturingsorgaan, waardoor wat dit orgaan betreft de instruktie als afgehandeld kan worden beschouwd. Dit kan nu normaal verder gaan met de volgende instrukties, terwijl het kanaal voor de verdere afhandeling van de betrokken in- of uitvoer zorgt draagt. Hiermee is de gewenste parallelwerking bereikt.

Gaan we iets nauwkeuriger kijken hoe het kanaal een dergelijke in- of uitvoer behandelt. We nemen daarbij weer het voorbeeld van het magneetband transport.
Van het magneetbandapparaat wordt continu een groep van 9 bits aangeboden, meer precies iedere 1/60000 sek=16,7 mikrosek. Het kanaal moet 6 van deze bitgroepen opvangen en samenstellen tot een woord. Dit is als volgt voor te stellen. Het kanaal bevat 2 registers, één van 6x9=54 bits en een tweede van 48 bits. In het eerste wordt steeds aan een zijde een groep van 9 bits ingebracht en daarna schuift alles 9 bits opzij. Na 6 keer is het register dan vol. Er kan nu pariteitskontrole plaatsvinden en wanneer dit in orde is worden alle 48 informatiebits parallel overgebracht naar het tweede register. Het eerste register kan weer opnieuw vollopen. Dit vollopen duurt 6x16,7=100 mikrosek.



Er moet dus voor gezorgd worden dat gedurende deze tijd er een keer gelegenheid is gevonden om de inhoud van het 2e register over te brengen naar de juiste plaats in het kernengeheugen. Met dit overbrengen zijn bovendien nog enkele andere werkzaamheden verbonden. Iedere keer moet n.l. de inhoud van het 2e register van het kanaal naar een

volgend adres in het geheugen worden overgebracht. Er moet dus een telling worden bijgehouden en het opbergen moet plaatsvinden aan de hand van deze telling.Deze telling moet bovendien iedere keer getest worden om na te gaan in hoeverre het transport klaar is en de magneetband weer moet worden afgeremd.
Voor al deze handelingen zijn natuurlijk een aantal elektronische schakelingen nodig.
In principe zijn deze schakelingen reeds aanwezig in reken- en besturingsorgaan en kan men hiervan gebruik maken.
In de praktijk vindt men deze werkwijze in het algemeen bij de goedkopere en langzamer werkende modellen rekenmachines.

De werkwijze is daarbij als volgt. Wanneer het 2e register in het kanaal weer is gevuld, wordt het lopende programma in de machine even opgehouden. De benodigde registers worden tijdelijk even vrijgemaakt door hun inhoud op te bergen in het geheugen. De zojuist beschreven handelingen worden geëffectueerd en daarna worden de registers weer in hun oude staat hersteld. Het programma loopt weer normaal verder. Dit alles wordt overigens volledig in het hardware van de machine gerealiseerd, zodat men hiervoor programmatisch geen enkele maatregel behoeft te nemen. Men betaalt echter wel op een andere manier een zekere prijs. Gebaseerd op de getallen in het gegeven voorbeeld duurt het opbergen van een woord nu niet 2 mikrosek maar een veelvoud hiervan. De werkelijk overblijvende tijd voor het programma wordt dus aanzienlijk gereduceerd.

Voor de sneller werkende rekenmachines vindt men dit in het algemeen een niet te aanvaarden oplossing en in dat geval worden in het kanaal compleet alle elektronische schakelingen opgebouwd om de benodigde handelingen te kunnen verrichten. De kanalen worden daarmee natuurlijk duurder.
Het gebruik van het geheugen wordt dan echter inderdaad beperkt tot de genoemde 2 mikrosek per op te bergen woord. Dit betekent dat het lopende programma max. 2 mikrosek per op te bergen woord wordt vertraagd.
Het woord max. is hier gebruikt want in vele gevallen zal het lopende programma geen enkele vertraging ondervinden. Tijdens de werkelijke uitvoering van een instruktie hebben reken- en besturingsorgaan het geheugen n.l. toch niet nodig. Alleen wanneer een nieuwe instruktie of een operand uit het geheugen moet worden opgehaald kan een konflikt situatie ontstaan als juist op dat moment ook een kanaal iets in het geheugen wil opbergen. Meestal is de machine dan zo gebouwd dat het kanaal prioriteit heeft boven het reken- en besturingsorgaan en deze organen dan één geheugencyclus moeten wachten.

Het is dus alsof af en toe het lopende programma een geheugencyclus wordt onderbroken en men spreekt dan wel van *"CYCLE STEALING"*

Of men nu de ene oplossing dan wel de andere of eventueel zelfs een tussenvorm kiest is een kwestie van economie bij het ontwerpen van de machine. Programmatisch gezien zijn deze oplossingen gelijk. Men dient daarbij ook goed in het oog te houden dat de hier genoemde programma-onderbreking niets te maken heeft met de nog hierna te noemen onderbreking die optreedt wanneer het totale transport klaar is.
Dan vindt er namelijk wel een totale programma-onderbreking plaats waarmee ook in de software rekening gehouden moet worden.

Zoals we aanstonds nog zullen zien moeten n.l. aan het eind van een totaal transport zoveel handelingen verricht worden dat het economisch niet meer verantwoord zou zijn dit alles nog in hardware in de kanalen in te bouwen.

Een kanaal bevat dus in het algemeen alle benodigde schakelingen om allereerst een gewenste transportopdracht in ontvangst te kunnen nemen, daarna het betrokken apparaat te kunnen starten, alle benodigde tellingen te kunnen bijhouden en ten slotte weer een gereedmelding te kunnen geven.
Daarenboven heeft het kanaal ook nog kontrolerende funkties, zoals de pariteitskontrole en moet er op ingericht zijn de nodige acties te kunnen ondernemen indien een eventuele fout optreedt.

Deze kanalen vormen dan ook vrij kostbare onderdelen van de machine en bij de keuze van een machineconfigurantie zal men zich ook terdege moeten bezinnen hoeveel van dergelijke kanalen men zal aanschaffen. Nu zijn deze kanalen wel zo gebouwd dat men hierop meerdere apparaten kan aansluiten, echter zijn de schakelingen slechts eenmaal aanwezig. Dit betekent dat een kanaal slechts één apparaat tegelijkertijd kan bedienen. Wanneer men meerdere apparaten dus absoluut tegelijkertijd wil laten werken dan moeten ze op verschillende kanalen worden aangesloten, die onderling volkomen onafhankelijk kunnen werken.
Wel kan de eerder genoemde conflikssituatie iets vergroot worden wanneer op een bepaald moment n.l. toevallig meerdere kanalen en het reken- of besturingsorgaan toegang tot het geheugen willen hebben.
In dat geval is er een ingebouwde voorrangsregeling waarbij de kanalen achter elkaar een beurt krijgen en daarna

het reken- en besturingsorgaan. De gezamelijke transportcapaciteit van de kanalen mag daarbij natuurlijk niet de snelheid van het kerngeheugen overschrijden.
In een dergelijk geval zou namelijk voor het laatste kanaal in de serie de situatie kunnen ontstaan dat het eerste register alweer is volgelopen voordat het 2e register overgeheveld kon worden naar het geheugen. Deze overwegingen gelden overigens alleen bij de keuze van de samenstelling van de machine op het moment van aanschaffing of bij uitbreiding van de installatie. Slechts bij zeer bijzondere installaties zal men hiermee ook tijdens het programmeren rekening moeten houden.

Tenslotte moeten we de kanalen nog in 2 typen onderscheiden. Zoals de beschrijving tot nu toe is gegeven heeft deze eigenlijk alleen betrekking op één type dat wel aangeduid wordt met de benaming "selector kanaal". Dergelijke typen kanaal geven n.l. aanleiding tot moeilijkheden wanneer men een groter aantal zeer langzaam werkende apparaten moet aansluiten. Zeer langzaam dan, vergeleken met bijvoorbeeld de snelheid van een magneetband.

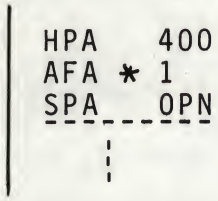
In het gegeven voorbeeld was voor de magneetband nog altijd sprake van 60000 karakters per sekonde. Wanneer men echter denkt aan een aantal elektrische schrijfmachines dan werkt men in de orde van 10 tot 15 karakters per sek. Zou men een aantal van dergelijke apparaten parallel willen laten werken dan zou dit praktisch er op neer komen dat men een even groot aantal selectorkanalen zou moeten installeren hetgeen tot onmogelijke kosten zou voeren. Speciaal voor dit soort langzaam werkende apparaten kent men daarom nog een ander type kanaal dat meestal aangegeven wordt met de benaming *"MULTIPLEX KANAAL"*.
Een multiplex kanaal is in staat tegelijkertijd een groot aantal langzaam werkende apparaten te bedienen. D.w.z. dat het multiplex kanaal per aangesloten apparaat informatie langzaam opzamelt tot een bepaalde informatie-eenheid, bijv. ter grootte van een woord van de machine, en dan deze informatie-eenheid weer doorgeeft aan het geheugen. Voor ieder apparaat kan daarbij dan deze informatie in verschillende gebieden in het geheugen worden opgeborgen. Vanuit programmatisch oogpunt is het dan alsof deze langzaam werkende apparaten toch via verschillende selectorkanalen zijn aangesloten. Daarmee is het meer een kwestie van andere technische uitvoering in verband met de economie van de machine. In de verdere beschouwing over de kanaalorganisatie speelt dit verschil dan ook geen rol.

Ir. D.H. Wolbers

INGREEP OF INTERRUPT

In de vorige paragraaf werd alleen gesproken over datgene wat gebeurt tijdens het transport van één blok gegevens. Hoewel het geheel wel via software in gang gezet wordt is de afhandeling toch meer een hardware kwestie.
Bij de beslissingen die genomen moeten worden tussen de verschillende bloktransporten naar en van diverse apparaten spelen zoveel faktoren een rol dat, zoals reeds eerder opgemerkt, dit niet meer rechtstreeks in hardware is op te lossen.
Enkele van deze faktoren willen we nader beschouwen en daarmee komen tot de logische oplossingen, die men in de praktijk hiervoor kiest.
In de eerste plaats bekijken we de konsekwenties van het verschil tussen in- en uitvoeropdrachten enerzijds en de andere opdrachten anderzijds.
Bij het samenstellen van een programma in een hele serie van opdrachten wordt er van uitgegaan, dat iedere opdracht volledig is afgehandeld voor de volgende aan de beurt komt. Voor de in- en uitvoeropdrachten gaat dit nu echter niet meer op. Daarom worden deze opdrachten ook op een betere manier aangegeven met startopdrachten voor in- of uitvoer. Bij het gebruik van deze opdrachten in een programma ontstaan nu moeilijkheden.
Onmiddellijk na een startinvoeropdracht mag men in programma bijvoorbeeld nog geen opdrachten geven die van de in te lezen informatie gebruik zouden moeten maken omdat op het moment dat deze opdrachten aan de beurt zijn voor uitvoering de gewenste informatie eenvoudig nog niet in het kerngeheugen staat.
Gedurende het transport mogen in het lopende programma dus alleen opdrachten voorkomen, die op geen enkele wijze betrekking hebben op de invoer.
Hoe bepaalt men nu het moment waarop het transport klaar is? Een eerste oplossing, die men bij enkele oudere machines wel heeft toegepast, is dat men tijdens het programmeren zich volledig rekenschap geeft van de tijdsduur van iedere uit te voeren opdracht alsmede van de tijd die nodig is voor het gewenste informatietransport. Bij ieder informatietransport doen zich dan 2 mogelijkheden voor.
Ten eerste, na het geven van de startopdracht zijn nog zoveel andere opdrachten gegeven, dat het transport zeker klaar is. Men kan daarna veilig opdrachten geven, die gebruik maken van de juist ingelezen informatie.
Het tweede geval doet zich voor wanneer eigenlijk te weinig opdrachten nog kunnen worden uitgevoerd. Men moet dan in het programma kunstmatig een zekere wachttijd introduceren. Dit is te realiseren door een klein lusprogram-

ma dat niets doet dan tellen en testen , b.v.



Nemen we eens aan dat de opdracht AFA een tijdsduur heeft van 15 mikrosek en de SPA opdracht 10 mikrosek. De kunstmatig geïntroduceerde wachttijd is dan 400x25=10000 mikrosek.
Dit is natuurlijk een weinig bevredigende oplossing, want van de gewenste parallelwerking is weinig nuttigs overgebleven.
Men moet bovendien ten aanzien van de tijdschattingen altijd aan de veilige kant blijven waardoor nog weer extra machinetijd verloren gaat. Daarenboven kunnen zich situaties voordoen, waarbij tijdschattingen niet eens meer uitvoerbaar zijn.
Bijvoorbeeld men geeft een startopdracht om een aantal regels op de regeldrukker af te drukken. Vlak voordat deze startopdracht aan het kanaal wordt overgegeven constateert echter de operateur dat het papier op is of het inktlint niet goed is. Hij schakelt daarom de regeldrukker even uit, brengt de nodige voorzieningen aan en schakelt weer in.
Het kanaal heeft de intussen ontvangen startopdracht vastgehouden en begint nu pas met de uitvoering. Het is duidelijk dat men met deze omstandigheden bij het samenstellen van een programma geen rekening heeft kunnen houden.
Nu kan men wel maatregelen bedenken, die ook in dit soort gevallen nog een oplossing geven, maar die voeren dan eigenlijk tot een andere oplossing van het tijdschattingsprobleem.

Het principe van deze oplossingen kunnen we als volgt omschrijven. Het kanaal bevat alle nodige schakelingen om de totale afloop van de gewenste in- of uitvoer te regelen en te kontroleren. Hier wordt dus op eenduidige wijze het preciese tijdstip geconstateerd waarop het transport klaar is. Men kan dan in ieder kanaal een eenvoudig elektronisch register inbouwen dat slechts twee standen kan innemen namelijk aan of af.
Daarbij wordt er voor gezorgd dat dit register aan staat zolang een informatietransport nog bezig is en pas wordt afgezet wanneer het klaar is.

In de totale opdrachten-set definiëren we bovendien enkele nieuwe testopdrachten.

Ir. D.H. Wolbers

Stel dat we de kanalen symbolisch aangeven met P,Q,R,S enz. dan zijn er ook een aantal overeenkomstige testopdrachten SKP, SKQ, SKR, SKS enz.
Al deze opdrachten kunnen we omschrijven als voorwaardelijke sprongopdrachten met als voorwaarde : Spring als betrokken kanaalregister aan staat.

Nemen we als voorbeeld dat we in een programma een startopdracht geven voor kanaal P. Een eindje verder in het programma willen we er zeker van zijn dat de daarop volgende instrukties pas worden uitgevoerd nadat het transport klaar is.
Dit is te bereiken met één instruktie namelijk :

```
OPN | SKP ; OPN
```

Men noemt dit wel een voorwaardelijke cirkelstop. De volgende instruktie wordt n.l. pas uitgevoerd als de test mislukt en dat is juist als het speciale kanaalregister weer is afgezet.
Men komt hiermee dus wel tegemoet aan het tijdschattingsprobleem maar van een werkelijke parallelwerking, althans een nuttige, is toch geen sprake. Bovendien moet men voor het realiseren van een dergelijk systeem toch ook nog wel andere maatregelen nemen. Stel dat men bijvoorbeeld in een programma een startopdracht geeft voor een apparaat dat toevallig tijdelijk is gestoord. In het programma komt dan even later de voorwaardelijke cirkelstop, maar aangezien het kanaal niet klaar komt blijft het kanaalregister aan staan en effektief komt daarmee de hele machine tot stilstand.
Nu is voor dergelijke situaties ook wel een oplossing denkbaar maar daarop willen we hier niet verder ingaan.
We zouden deze methode overigens enigszins kunnen omschrijven als een methode met passieve gereedmelding van het kanaal. Het passieve bestaat dan hieruit dat het kanaal feitelijk alleen maar een melding geeft van het gereedkomen van een transport, terwijl men in het programma hierop pas reageert met behulp van de speciale voorwaardelijke sprongopdracht.

We kunnen nog een stapje verder gaan en van een passieve gereedmelding overstappen naar een methode met actieve gereedmelding, waarbij na het gereedkomen van een transport ook onmiddellijk een bepaalde actie zal plaatsvinden. Dit is dan bovendien de methode die in vrijwel alle moderne machines wordt toegepast.
En waaruit bestaat dan deze actieve handeling? Beter is het eigenlijk te spreken over een hele keten van handeling-

en die verricht moeten worden.
Enkele daarvan kunnen we reeds overzien. Zo zullen we ergens willen registreren dat een bepaald transport klaar is. Juist omdat in- en uitvoerapparaten relatief langzaam werken is het zaak deze zo effectief mogelijk te gebruiken en bij name een kanaal niet ongebruikt te laten wanneer er nog in- of uitvoerwerkzaamheden moeten worden uitgevoerd. Indien enigszins mogelijk zal men na het klaarkomen van een transport zo snel mogelijk weer de volgende startopdracht willen geven.
Deze en nog een aantal andere handelingen die we aanstonds leren kennen, zullen moeten worden uitgevoerd en daarvoor is het nodig dat de machine tijdelijk zich zal bezighouden met een speciaal programma waarin al deze aktiviteiten kunnen worden gerealiseerd. Dit betekent dat het lopende programma met alle nodige voorzorgen tijdelijk moet worden onderbroken.
Deze onderbreking is van geheel andere aard dan de onderbreking zoals genoemd in de vorige paragraaf tijdens een informatietransport. Daarbij ging het alleen om "cycle stealing" of eventueel om het gebruik van een enkel register dat dan veilig gesteld moest worden.
Nu gaat het echter om een complete onderbreking van het programma en dan op zodanig manier, dat het later weer zonder fouten kan verder gaan vanaf het punt waar het was onderbroken.

Men dient zich bovendien te realiseren dat niet voorzien kan worden op welk punt in het programma deze onderbreking zal optreden. Dit betekent dat de onderbreking technisch zo gerealiseerd moet worden dat de inhoud van alle registers, die mogelijkerwijs door het programma in gebruik zijn, veilig gesteld worden zodanig dat ze later weer in de oude staat kunnen worden hersteld.

INTERRUPT

Daarmee komen we tot de wijze waarop de actieve gereedmelding van een kanaal geëffectueerd wordt, n.l.een op veilige wijze onderbreking van het lopende programma. Deze faciliteit staat bekend onder de naam *ingreep* of *interrupt*. De ingreep zelf is dus een hardware kwestie en moet bij de konstruktie van de machine reeds worden ingebouwd.

Onderbreken van een programma betekent echter primair ingrijpen in de gang van zaken in het besturingsorgaan. We moeten ons de gang van zaken bij de ingreep dan ook als volgt voorstellen. Zodra een kanaal klaar is met een bepaald transport vindt er elektronisch een melding

plaats aan het besturingsorgaan.
Zoals we vroeger hebben gezien werkt het besturingsorgaan in 3 fasen.
Ten eerste ophalen van de volgende instruktie met behulp van de inhoud van de opdrachtenteller.
Ten tweede uitvoering van de instruktie, zoals die in het uitvoeringsregister staat.
Ten derde ophogen van de opdrachtenteller.
Dat men in veel gevallen om snelheid te winnen de tweede en derde fase vaak gelijktijdig laat plaatsvinden geeft een principiële wijziging. In ieder geval na de derde fase begint het besturingsorgaan weer met de eerste.
Bij een machine ingericht met ingreep is er echter een vierde fase. In deze vierde fase wordt alleen gekeken of er een melding van een of ander kanaal is aangekomen.
Zo niet dan wordt normaal vervolgd met de eerste fase en gaat dus het lopende programma gewoon door. Is er echter gedurende de eerste drie fasen wel een melding binnen gekomen van een kanaal dan gebeurt er in de vierde fase iets anders.
We kunnen deze vierde fase dus als het werkelijke moment van ingreep markeren. Meestal drukt men dit zo uit dat bij ingreep de lopende instruktie eerst nog wordt afgehandeld. Wat er althans in hardware tijdens deze vierde fase plaats vindt hangt af hoe snel men deze ingreep wil laten verlopen en hoeveel kosten men daarvoor bij de construktie over heeft.
Minimaal moet het volgende in ieder in hardware gerealiseerd worden. De inhoud van de opdrachtenteller wordt opgeborgen op een van te voren vastgestelde plaats in het geheugen. Daarmee wordt dus bewaard het punt waarop het lopende programma is onderbroken. Tevens wordt de opdrachtenteller gevuld met een ander te voren vastgesteld getal. Men kan hiermee volstaan en het besturingsorgaan weer normaal aan de eerste fase laten beginnen. Dit betekent echter dat niet de volgende instruktie van het lopende programma wordt opgehaald, maar een instruktie die staat op het adres dat door de ingreep expliciet in de opdrachtenteller is ingevuld.
De ingreep manifesteert zich dus als een geforceerde spronginstruktie, waarbij een soort terugkeeradres in veiligheid is gebracht. Het hangt volledig af van de opgehaalde instruktie en de daarop volgende instrukties, wat er precies zal gebeuren. Al deze instrukties vormen tezamen een onderdeel van een speciaal programma, dat we hier verder zullen aanduiden met de naam coördinator. Elders in de literatuur komt men als namen hiervoor ook wel tegen *coördinatorprogramma, monitor, monitorprogramma, supervisor,* e.d.

We zullen nu deze coördinator iets nader beschouwen en aangezien het hier een programma betreft kunnen we dit ook weer in volledige software terminologie beschrijven. Omdat tijdens de werking van de coördinator ook reken-registers e.d. nodig zijn begint het programma met het in veiligheid brengen van deze registers. In SERA zouden dat b.v. zijn A en B accumulator TERUG en ALARM register. Dit vergt het uivoeren van een aantal instrukties en kost dus een zekere tijd. Deze tijd is te bekorten indien bij de ingreep zelf al deze registers reeds automatisch worden opgeborgen, maar dit vergt meer hardware waardoor de machine construktief duurder wordt. Overigens is deze keus beperkt tot die registers die qua software bereikbaar zijn.
In sommige machines bestaan er soms hardware registers, die ook na afloop van een instruktie nog informatie bevatten zonder dat men die expliciet in het programma als zodanig herkent. Het eerste voorbeeld daarvan is de reeds besproken opdrachtenteller. Bij een machine als SERA moet men daarbij ook denken aan het modificatieregister.

Wanneer in het lopende programma voorkomt

```
MOD   TEL
HPA   LYST
```

dan is mogelijk dat juist een ingreep optreedt na uitvoeren van MOD opdracht en voor uitvoering van HPA.
Op dat moment is het modificatieregister dus gevuld. In de eerste plaats moet de inhoud hiervan dan bewaard worden en in de tweede plaats moet het register op 0 gezet worden omdat anders de eerste instruktie van de coördinator gemodificeerd zou worden met een grootheid uit het lopende programma. Dit betekent dat in een machine als SERA het modificatieregister dan ook behoort tot de groep die in ieder geval door de hardware gered wordt. Nadat op deze wijze alle registers veilig gesteld zijn komen in de coördinator de instrukties aan bod, die de verschillende taken van de coördinator realiseren.
De eerste voor de hand liggende taak is de registratie van deze ingreep, die we kunnen opvatten als een gereedmelding, en hoe kunnen we daar nuttig gebruik van maken?

Daarmee komen we aan de funktie van de coördinator in het algemeen. Zeer globaal kan men die als volgt omschrijven.

De coördinator is een programma, dat de organisatie van alle in- en uitvoer beheerst, daarvoor de nodige administratie bijhoudt en wanneer nodig kontrolerend en regelend optreedt.

Ir. D.H. Wolbers

Zoals hiervoor besproken wordt iedere in- en uitvoer ingeleid door een daartoe strekkende startopdracht en eindigt met een ingreep. Bij dit laatste gebeuren wordt de coördinator automatisch ingeschakeld.

Om het geheel te kunnen beheersen moet ook echter het begin via de coördinator plaatsvinden. Daarom wordt er voor gezorgd dat alleen in de coördinator startopdrachten voor in- en uitvoer voorkomen en in normale programma's mogen deze niet gebruikt worden.

Om mogelijke fouten te vermijden wordt er zelfs voor gezorgd dat deze startopdrachten in een normaal programma zelfs niet gebruikt kunnen worden.
Dit is op de volgende manier mogelijk. Alle instrukties die in de machine zijn ingebouwd worden verdeeld in twee groepen, normale instrukties en beschermde instrukties. Verder kan het besturingsorgaan verkeren in 2 verschillende toestanden, veelal ook aangeduid als modes, de normale mode en de coördinatormode.
Wanneer het besturingsorgaan verkeert in de coördinatormode dan wordt iedere aangeboden instruktie zonder meer uitgevoerd. Is echter het besturingsorgaan in de normale mode, dan worden alleen normale instrukties gewoon uitgevoerd terwijl bij aanbieding van een beschermde instruktie niet tot uitvoering wordt overgegaan.

Tot de groep van de beschermde instrukties behoren nu alle startopdrachten voor in- en uitvoer. Tevens wordt er voor gezorgd dat gedurende alle tijd dat een gewoon programma wordt uitgevoerd het besturingsorgaan in de normale mode werkt. Daarmee is het voor een dergelijk programma dan onmogelijk geworden om zelf nog startopdrachten te laten uitvoeren. Dit betekent niet dat daarmee het initiatief voor een in- of uitvoer niet van een dergelijk programma zou kunnen uitgaan. Men is echter gedwongen daartoe de coördinator in te schakelen. Dat vindt dan plaats door middel van een speciale sprongopdracht.

Deze speciale sprongopdracht heeft 3 dingen tot gevolg.
Ten eerste wordt gesprongen naar eenbepaald punt in de coördinator.
Ten tweede wordt de inhoud van de opdrachtenteller vermeerderd met 1, veilig gesteld in een bijzonder register. Dit is dus te vergelijken met de werking van de SSP opdracht alleen wordt in dit geval niet het register TERUG gebruikt voor het terugkeeradres.

Ten derde wordt het besturingsorgaan omgeschakeld van normale mode naar coördinatormode, waarmee aan de coördinator wel de mogelijkheid gegeven wordt de startopdracht te laten uitvoeren. In het gewone programma heeft een dergelijke sprongopdracht dus meer het effect van een startwens. Om deze reden komt bij sommige machines het idee van sprongopdrachten ook niet meer tot uiting in mnemotechnische benamingen, maar worden deze veel meer gekozen in de richting van het uiteindelijke effect. Als zodanig moeten we dan ook beschouwen de eerder geleerde magneetbandopdrachten zoals MVS, MVL enz.

Deze opdrachten roepen dus op heimelijke wijze de coördinator aan, die verder programmatisch uitzoekt of de impliciet aangevraagde startopdracht gerealiseerd kan worden. Zo ja, dan wordt deze ook gegeven, zo nee dan wordt deze aanvraag voorlopig op een wachtlijst geplaatst, die we verder in de beschrijving zullen aanduiden met de naam startlijst. Daarna keert de coördinator aan de hand van het speciale register weer terug naar het programma dat de startwens te kennen had gegeven. Dit laatste gebeurt dan overigens ook weer met een speciale sprongopdracht, die naast het effect van de normale sprong er voor zorgt dat het besturingsorgaan weer teruggeschakeld wordt naar de normale mode.

Voor het gewone programma hebben opdrachten zoals MVS, MVL enz. ogenschijnlijk dus het effect van een startopdracht, waarom men ze ook wel aanduidt met de benaming pseudoinstrukties. Men dient daarbij wel in het oog te houden dat deze pseudoinstrukties van een ander type zijn als BGN, SRT en VRY. De laatst genoemde zijn alleen aanwijzingen voor de assembler en hebben hun betekenis na het inlezen van het programma verloren. De andere pseudo-opdrachten worden door de assembler vertaald in de genoemde speciale sprongopdrachten.

We hebben nu reeds 2 situaties leren kennen waarbij de coördinator wordt ingeschakeld. Ten eerste wanneer een lopend programma een startwens heeft voor in- of uitvoer. Ten tweede na het optreden van ingreep. Voor het geval van ingreep kunnen we daarbij vermelden dat ook dan automatische omschakeling naar coördinatormode plaatsvindt. Dit geldt verder algemeen.
Er wordt steeds voor gezorgd dat tijdens de werking van de coördinator het besturingsorgaan in de coördinatormode verkeert en in de andere gevallen in de normale mode.

Wel moeten we nog een andere eigenschap van de coördinator-

mode vermelden. Deze eigenschap is nodig opdat de coördinator op de juiste manier zijn regelende funktie kan uitvoeren. Daartoe denken we aan de volgende situatie. Aan de machine kunnen meerdere kanalen verbonden zijn, tegelijkertijd kunnen daardoor een aantal informatietransporten plaats hebben. Op een gegeven moment komt een van deze transporten klaar en dit leidt tot een ingreep. Terwijl dan de coördinator bezig is de gevolgen hiervan te verwerken kan een ander transport klaar komen. Dit zou dan weer tot ingreep voeren. Dit zou zelfs meerdere keren over elkaar heen kunnen plaatsvinden en het is nu wel voor te stellen dat er dan onontwarbare puzzels ontstaan. Deze situatie is als volgt te verhinderen. Wanneer het besturingsorgaan werkt in de coördinatormode wordt niet meer gereageerd op ingreep. Het besturingsorgaan slaat dan eigenlijk de vierde fase, waarin kanaalmeldingen werden afgetast, over. Daardoor bereikt men dat bij min of meer gelijktijdig optreden van kanaalgereedmeldingen de daarbij behorende ingrepen stuk voor stuk kunnen worden afgehandeld.

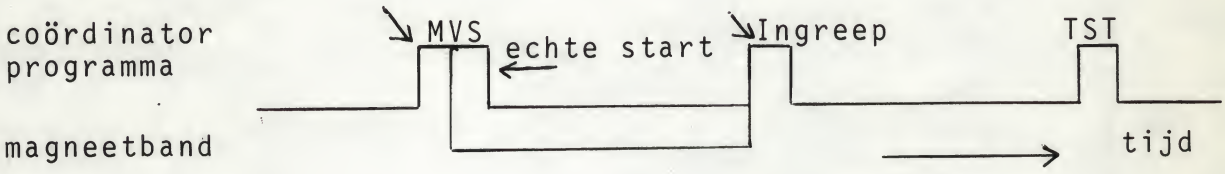
In vele gevallen zal de coördinator met deze gereedmeldingen direkt nog niet veel kunnen beginnen en deze worden dan bewaard in een speciale lijst, die we hierna als bloklijst zullen aanduiden. Deze wordt dan speciaal geraadpleegd in een derde situatie waarbij de coördinator te hulp wordt geroepen. Immers, wanneer in een gewoon programma een startwens gegeven is, dan is daarin verder niet meer te overzien wanneer het daarmee beoogde informatietransport klaar is. Men gaat daarom als volgt te werk. Na het geven van een of meerdere startwensen worden nog zoveel instrukties uitgevoerd als maar mogelijk is zonder dat daarbij het gewenste informatietransport een rol speelt.
Is men in het programma op een punt aangekomen dat men zeker moet zijn dat de gevraagde in- of uitvoer klaar is dan vraagt men dit programmatisch aan de coördinator. Dit gebeurt ook weer door een speciale sprong naar de coördinator, waarbij men echter binnenkomt op een ander punt dan in het geval van een startwens. Ook hier brengt men het effect van deze sprongopdracht vaak weer tot uitdrukking in de mnemotechnische benaming.
In SERA hebben we daarvoor bij de behandelde magneetbandopdrachten leren kennen de opdracht TST. Constateert de coördinator dan aan de hand van de bloklijst dat het gevraagde transport klaar is dan wordt weer teruggesprongen naar de opdracht direkt volgend op de TST opdracht.

Er rest dan de vraag, wat gebeurt er na een TST opdracht als de gereedmelding nog niet in de bloklijst staat. Men kan zich op dit punt gekomen zelfs afvragen, waarom een

dergelijk schijnbaar ingewikkeld systeem gekozen wordt voor in- en uitvoer, dat naast een aantal extra hardware voorzieningen bovendien nog een ingewikkeld programma in de vorm van de coördinator vereist.

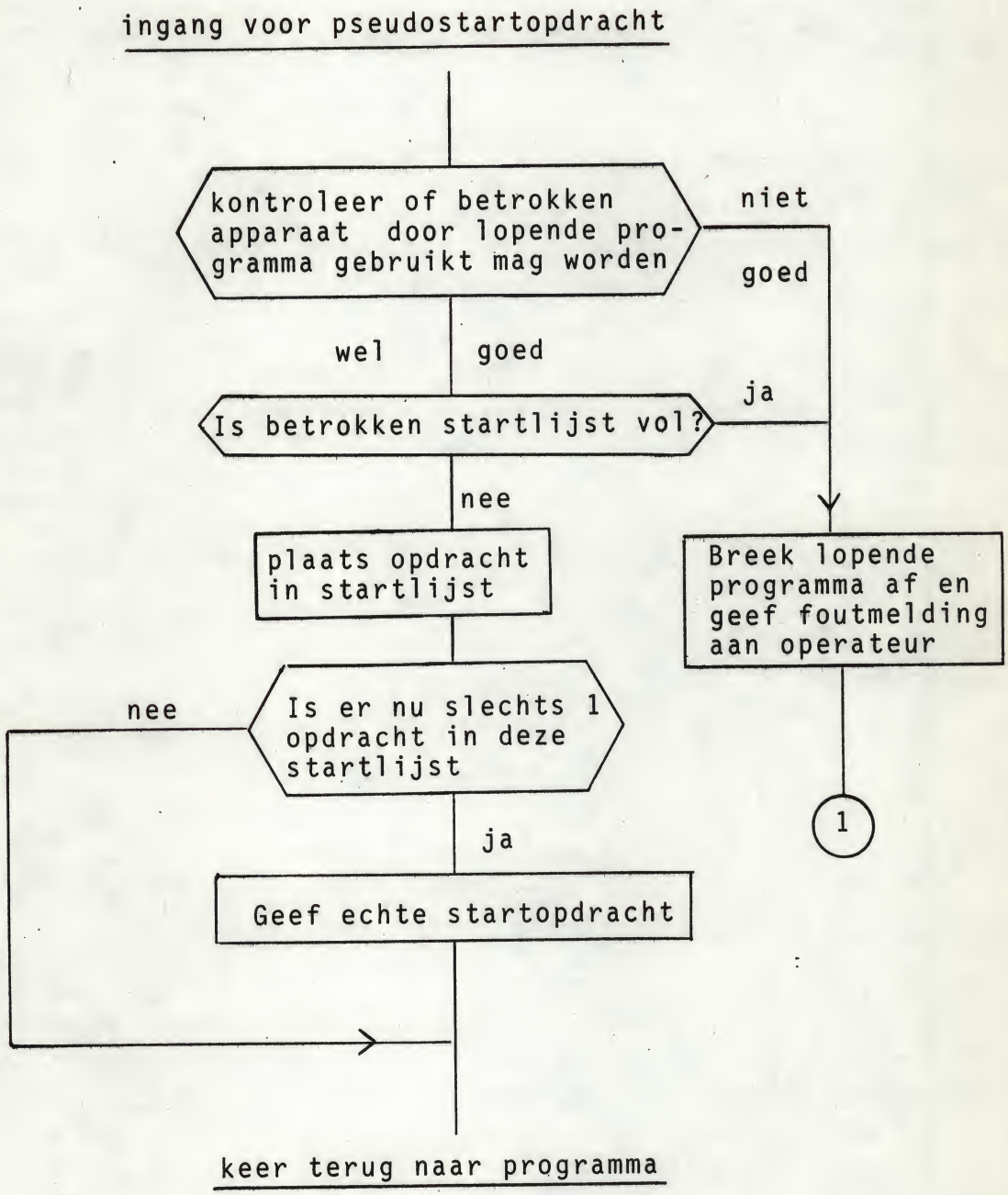
Daartoe geven we de hoofdlijnen van de coördinator in de vorm van 3 grove stroomschema's en gaan de werking hiervan na aan de hand van enkele voorbeelden.

We beginnen met een eenvoudig geval en beschouwen dat in een SERA programma een opdracht MVS gegeven wordt en enige tijd later de bijbehorende TST opdracht, waarbij het zo uitkomt,dat het transport naar de magneetband reeds klaar is voor de TST opdracht optreedt.
Het gehele gebeuren kunnen we dan als volgt in tijdschema weergeven.



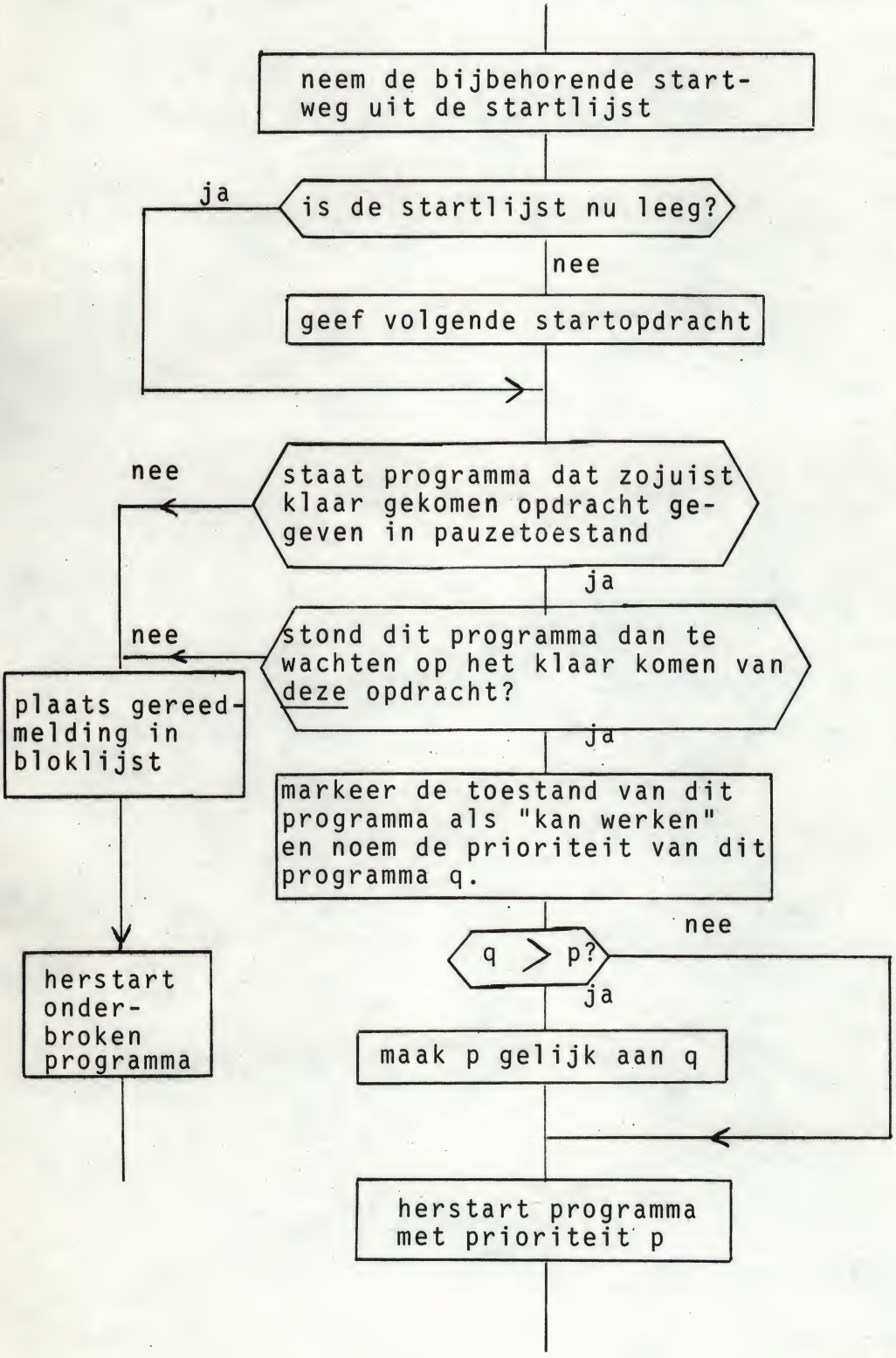
In horizontale richting geven we schematisch de tijd aan. Daarbij is niet mogelijk over het gehele gebied dezelfde tijdschaal toe te passen omdat b.v. de tijd, dat de coördinator werkt in de praktijk veel kleiner is dan de tijd benodigd voor het transport naar de magneetband.

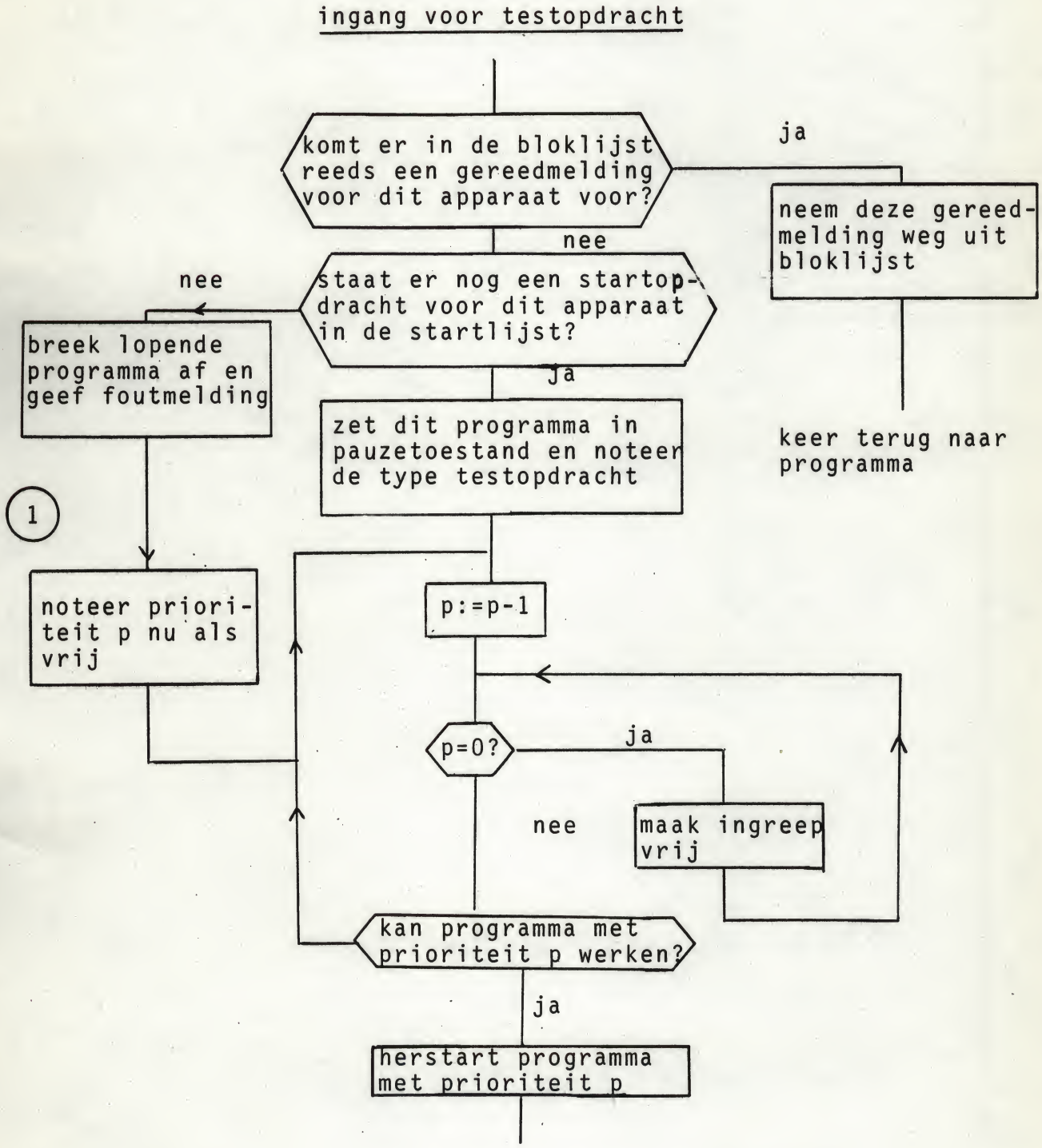
Ir. D.H. Wolbers



Ir. D.H. Wolbers

Ingang na ingreep waarbij aangenomen dat op technische wijze alle registers reeds in veiligheid gebracht zijn.

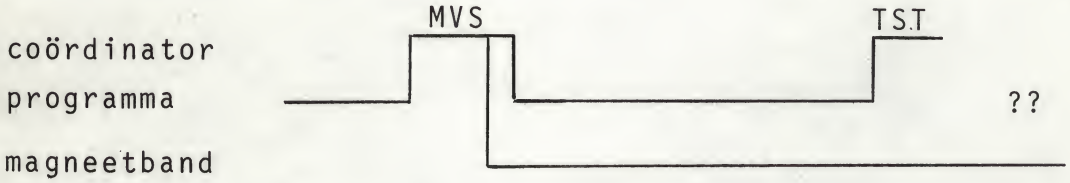



ingang voor testopdracht
komt er in de bloklijst reeds een gereedmelding voor dit apparaat voor?
ja
neem deze gereed-melding weg uit bloklijst
keer terug naar programma
nee
staat er nog een startop-dracht voor dit apparaat in de startlijst?
nee
breek lopende programma af en geef foutmelding
ja
zet dit programma in pauzetoestand en noteer de type testopdracht
1
noteer priori-teit p nu als vrij
p:=p-1
p=0?
ja
nee
maak ingreep vrij
kan programma met prioriteit p werken?
ja
herstart programma met prioriteit p

Men moet het schema als volgt interpreteren. De computer voert achtereenvolgens instrukties uit van het gegeven programma. Daarin komt op een gegeven ogenblik voor MVS. Daardoor gaat de coördinator werken en het programma staat stil. Na het doorlopen van enkele instrukties geeft de coördinator de echte startopdracht. Vanaf dat moment wordt ook de magneetband gebruikt. De coördinator voert nog enkel instrukties uit en keert dan weer terug naar het programma. Nu treedt een periode op dat 2 dingen gelijk plaats vinden.
Het besturingsorgaan voert instrukties van het programma uit en de magneetband krijgt informatie toegevoerd via het kanaal. Dit gaat door tot het moment waarop het transport klaar is. Dan volgt ingreep. Daardoor wordt het lopende programma onderbroken en de coördinator komt weer aan bod. Deze constateert dat het programma nog niet om het klaarkomen heeft gevraagd en plaatst de gereedmelding in de bloklijst. Daarna wordt teruggekeerd naar het programma, waarbij alle registers weer in de oude staat worden teruggebracht, zoals die was op het moment van de ingreep. Tenslotte komen we in het programma aan de TST opdracht. Wederom komt de coördinator in werking. Deze constateert de gereedmelding in de bloklijst, zorgt dat deze melding hieruit verwijderd wordt en keert terug naar het programma.

Als volgend voorbeeld kiezen we bijna het zelfde geval echter met dit verschil dat nu de TST opdracht reeds gegeven wordt voordat het transport klaar is.

Dan ontstaat de volgende situatie

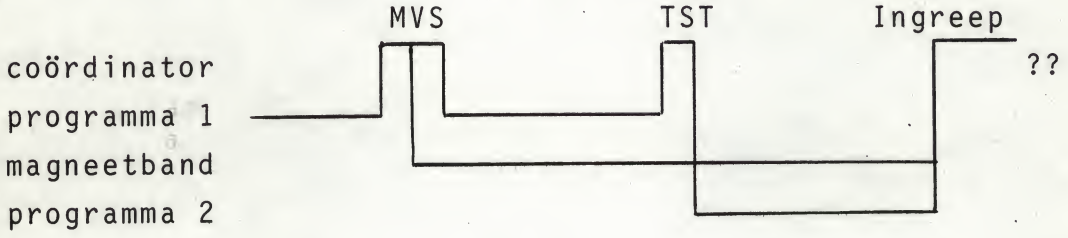


Met vraagtekens is daarbij de probleem situatie voor de coördinator aangegeven. Deze constateert uit het ontbreken van de gereedmelding in de bloklijst dat nog niet naar het programma teruggesprongen mag worden. Besturingsorgaan en rekenorgaan kunnen dus niet meer voor het programma gebruikt worden. Dit betekent dat we de hiermee noodzakelijkerwijs geïntroduceerde wachttijd kunnen benutten voor een ander programma, mits dit althans ook in het kernengeheugen aanwezig is.
Op deze wijze kunnen onderling onafhankelijke programma's elkaars wachttijden opvullen, terwijl bij een goede organisatie bovendien bereikt kan worden dat alle langzame

randapparaten zo optimaal mogelijk worden gebruikt.
Het systeem is ook niet beperkt tot 2 programma's want ook het tweede programma kan op een gegeven ogenblik moeten wachten op in- of uitvoer zodat dan het derde programma aan de beurt komt enz.

Het toepassen van een dergelijk programmeringssysteem noemt men *multiprogrammering*. Daarbij dient men goed in het oog te houden dat wel meerdere programma's tegelijkertijd in geheugen staan, maar dat op ieder moment slechts één programma aan het werk is.
Wel kunnen tegelijkertijd naast de werking van een programma een aantal in- of uitvoerapparaten bezig zijn, waarbij de mogelijke gelijktijdigheid hoofdzakelijk bepaald wordt door het aantal kanalen.
Tengevolge van de eis dat alle programma's gelijktijdig in het geheugen moeten staan is er wel een praktische begrenzing aan het aantal programma's, dat in multiprogrammering kan werken.
Heel globaal kan men stellen dat een efficiënter gebruik van besturingsorgaan, rekenorgaan en randapparaten moet worden afgewogen tegen de hogere kosten voor het benodigde geheugen.

We vervolgen nu het laatste voorbeeld en nemen even aan dat het tweede programma niet om in- of uitvoer vraagt en komen dat tot de volgende probleemsituatie.



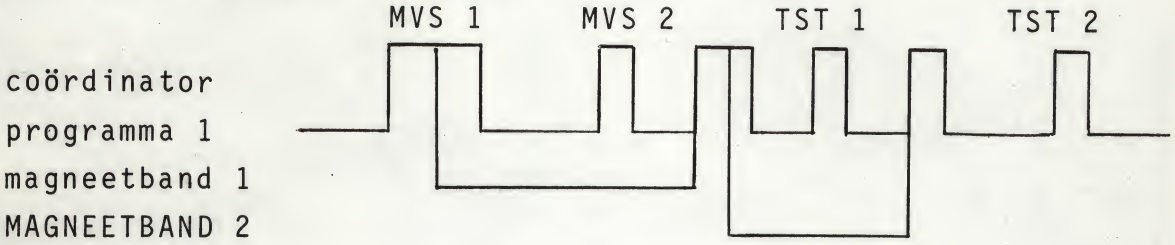
Terwijl programma 2 werkt komt het magneetbandtransport klaar waardoor ingreep volgt en de coördinator weer werkt. Vanaf dit moment zou dus programma 1 weer verder kunnen werken maar ook programma 2.
Naar welk programma moet de coördinator nu terugspringen? Het criterium dat men daarvoor wil aanleggen wordt niet meer beheerst door bepaalde technische noodzaken maar is veeleer een kwestie van strategie.
Een mogelijkheid zou zijn in het gegeven voorbeeld gewoon door te gaan met programma 2. Wanneer dan dit programma op een gegeven moment moet wachten op gewenste in- of uitvoer, over te schakelen naar een derde programma. Stel dat er maar 3 programma's zijn dan kan men op het moment dat programma 3 niet verder kan werken weer beginnen met 1 enz.

Op deze wijze komen dan alle programma's achter elkaar steeds weer aan bod. Deze strategie wordt slechts zelden toegepast. Het nadeel is n.l. dat een programma dat relatief veel "rekent" de machine vrijwel gaat beheersen terwijl een programma dat zeer weinig rekent maar bijna alleen in- of uitvoer heeft, constant in de wachtstand komt te staan.
Dat laatste is nog niet zo erg, maar wanneer een programma dat zeer sterk in- en uitvoer gebonden is in de wachtstand komt te staan zonder daarbij de kans te hebben de benodigde in- en uitvoerapparaten te laten werken komen ook deze onderdelen vrijwel stil te staan.
De meest gevolgde strategie is dan ook, dat programma's met veel in- en uitvoer en weinig rekenwerk zoveel mogelijk de kans krijgen om te werken terwijl rekenintensievere programma's dan maar de overblijvende tijd moeten opvullen.
Deze werkwijze kan men bereiken door prioriteiten in te voeren, waarbij de programma's onderling verschillende prioriteit hebben. Steeds wanneer de coördinator dan een keus moet maken uit 2 of meer programma's, wordt van alle programma's die op dat moment zouden kunnen werken, dus niet in de wachtstand staan,diegene gekozen die de hoogste prioriteit heeft. De coördinator kent de prioriteit van ieder programma en weet ook wat de prioriteit is van het programma dat het laatst onderbroken is of de coördinator heeft aangevraagd.
Door vergelijking is de keuze dan te maken. In de stroomschema's voor de coördinator wordt de prioriteit van het op dat moment lopende programma met p aangegeven.

In het laatste voorbeeld zal dan teruggekeerd worden naar programma 1, aannemend dat die een hogere prioriteit heeft dan programma 2.
Multiprogrammering biedt dus de mogelijkheid de verschillende delen van de computer efficienter te gebruiken. Er worden echter ook enkele nieuwe problemen geïntroduceerd. In het volgende hoofdstuk zal nog ter sprake komen het probleem van de geheugenbescherming, hier willen we er op wijzen, dat via de coördinator ook kontrole moet worden uitgeoefend dat verschillende programma's niet door elkaar dezelfde apparaten gaan gebruiken. Men kan b.v. niet uitvoer naar een regeldrukker afwisselend door verschillende programma's laten plaats vinden, want dan komen op een blad papier de resultaten van geheel verschillende problemen compleet door elkaar te staan. Gedurende de duur van een programma zijn bepaalde apparaten dan ook als het ware toegewezen aan een bepaald programma.

Deze toewijzing zelf is geen taak meer van de coördinator net zo min als de toewijzing van de prioriteit. In de praktijk vinden deze toewijzingen voor middelgrote machines plaats door de operateur, voor de grotere systemen is er nog weer een overkoepelend programma dat deze zaken regelt.
Voor deze programma's vindt men verschillende namen zoals *bedrijfssysteem, operating systeem, scheduling supervisor, etc.*
Voor de coördinator geldt in het algemeen alleen de kontrolerende funktie. Constateert daarbij de coördinator een fout, die in het algemeen te wijten is aan een programmeerfout in het gewone programma, dan wordt het lopende programma definitief onderbroken en meestal wordt via een standaarduitvoermedium een foutboodschap uitgegeven, waaruit de programmeur kan begrijpen welke foute situatie is opgetreden.

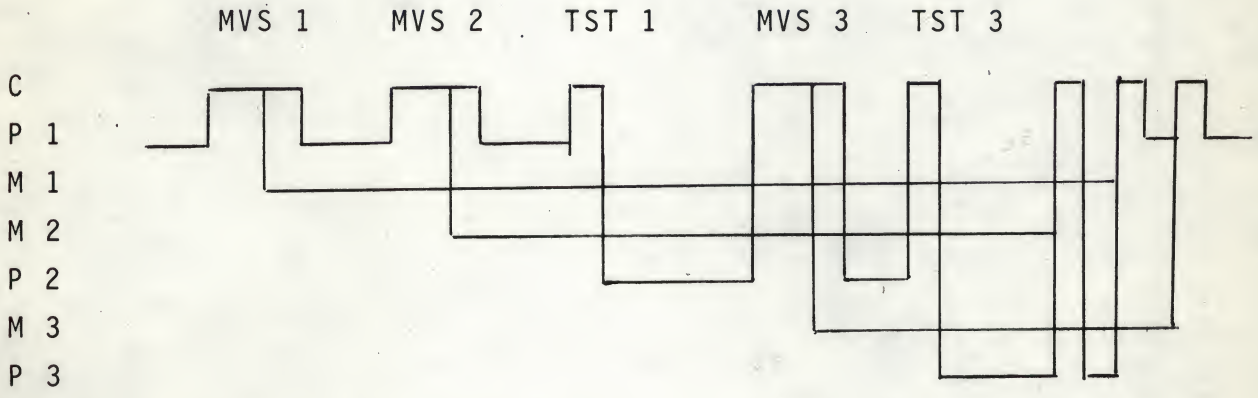
Enkele andere onderdelen van de coördinator worden nog iets duidelijker wanneer we de laatste 2 voorbeelden aangeven. Daarbij allereerst een eenvoudig geval waarbij in een programma kort na elkaar 2 startwensen voor 2 apparaten gegeven worden waarbij deze apparaten echter wel via hetzelfde kanaal zijn aangesloten zodat ze niet gelijktijdig kunnen werken.



In dit voorbeeld leidt de tweede MVS-opdracht dus niet direkt tot een startopdracht, deze wordt overigens wel zo snel mogelijk gegeven nadat dit mogelijk is, n.l. na de gereedmelding van magneetband 1.

Als laatste een iets ingewikkelder situatie met 3 programma's P 1, P 2 en P 3. Daarbij aangenomen P 1 als hoogste prioriteit en P 3 als laagste. Verder gebruikt P 1 de magneetbanden M 1 en M 2 via verschillende kanalen en P 2 gebruikt M 3 via een derde kanaal.

Ir. D.H. Wolbers



In dit voorbeeld vindt men de situatie waar P 1 staat te wachten op het klaarkomen van M 1 maar voordat dit plaats vindt komt M 2 klaar, die wel voor P 1 gewerkt heeft maar waarop P 1 niet stond te wachten. Bij een TST opdracht moet dus ook worden aangegeven op welk transport de test betrekking heeft.
Verder doet zich via dit voorbeeld de situatie voor dat een apparaat in dit geval M 3 gestart is door een programma van lagere prioriteit P 2 dat wel via ingreep een programma van hogere prioriteit P 1 even kan onderbreken maar de regie wordt daarna toch weer teruggegeven aan het programma met de hoogste prioriteit P 1.

Tenslotte nog enkele algemene opmerkingen.
Bij het ontwerpen van een coördinator moet men wel rekening houden met de situatie waarin toevallig toch geen enkel programma kan werken omdat alle aanwezige programma's moeten wachten. In het gegeven stroomschema is deze situatie zo opgelost dat dan blijkt dat de prioriteit P daalt tot 0. In dat geval wordt de ingreep weer vrijgegeven en de coördinator draait rond in een soort cirkelstop, waardoor natuurlijk besturingsorgaan en rekenorgaan dan niet meer nuttig worden gebruikt.
Aan deze toestand komt dan vanzelf een einde zodra er een ingreep volgt en dit een gereedmelding is van een transport waarop een der programma's stond te wachten ; dit programma komt dan vanzelf aan de beurt.

Zoals reeds opgemerkt is, is het hier gegeven schema van de coördinator slechts een grof blokschema. In werkelijkheid wordt het programma behoorlijk gecompliceerd door allerlei nevenfunkties. Zo geeft een kanaal niet altijd zonder meer een gereedmelding van het klaarkomen, maar ingreep kan ook plaatsvinden bij optreden van een technische fout van het betrokken in- of uitvoerapparaat. In sommige gevallen is daar programmatisch nog iets aan te doen. Bijv. wanneer bij vooruitlezen van een magneet-

band een pariteitsfout wordt geconstateerd kan een opdracht gegeven worden een blok terug te lezen en daarna opnieuw een blok vooruit. Dit zijn funkties die in de regel ook door de coördinator worden verzorgd. Dit gebeurt overigens maar een beperkt aantal keren b.v. 10 keer, omdat dan aangenomen wordt dat het een onherstelbare fout is. Daarvoor moet dus een telling worden bijgehouden, enz.

Een andere funktie van de coördinator houdt verband met de beschermde instrukties. Reeds is vermeld dat tijdens de werking van een gewoon programma beschermde instrukties niet worden uitgevoerd. Wat er in zo'n geval wel gebeurt, is dat er ook een ingreep volgt waardoor de coördinator weer werkt en kan constateren dat het lopende programma een fout heeft gemaakt.

Bij vele machines voeren ook programmatische onregelmatigheden zoals overloop, delen door 0 enz tot ingreep.
Ook een instruktie zoal STP voert bij dergelijke machines niet meer tot een echte stop van de machine maar betekent een sprong naar de coördinator waardoor het programma zichzelf in de wachtstand plaatst. De coördinator wordt daarmee het centrale programma die alle bijzondere situaties opvangt. De omvang mag dan ook niet onderschat worden. Een coördinator van 10000 instrukties is een vrij normale zaak.

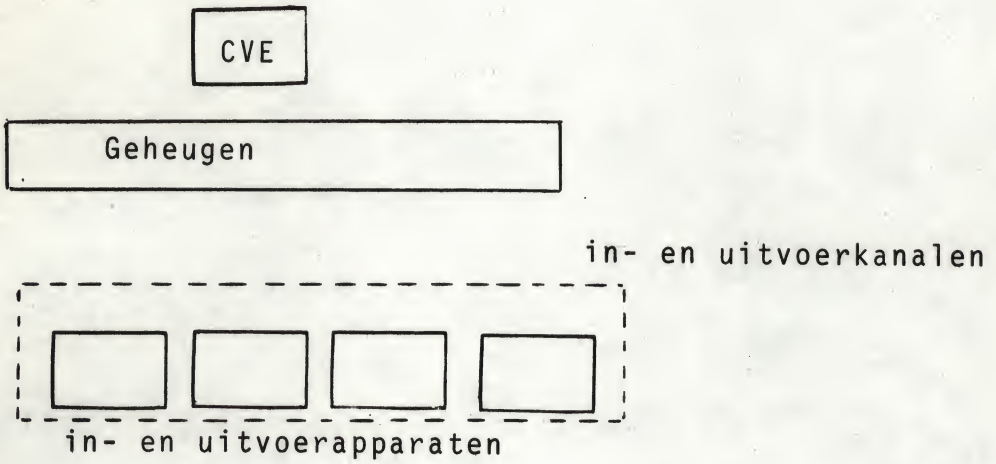
Tot zover een beschrijving van een multiporgrammeringssysteem. Helaas vindt men in de literatuur dergelijke systemen ook wel met andere namen aangegeven zoals multiprocessing en timesharing.Deze worden dan gebruikt omdat de werking van een dergelijke machine naar buiten de indruk geeft alsof allerlei verschillende processen tegelijkertijd werken.In navolging van het meest gangbare willen we het hiervoor beschreven systeem echter blijven aangeven met multiprogramming, terwijl de twee andere namen gebruikt zullen worden voor de hierna te beschrijven mogelijkheden.

CENTRALE VERWERKINGSEENHEID

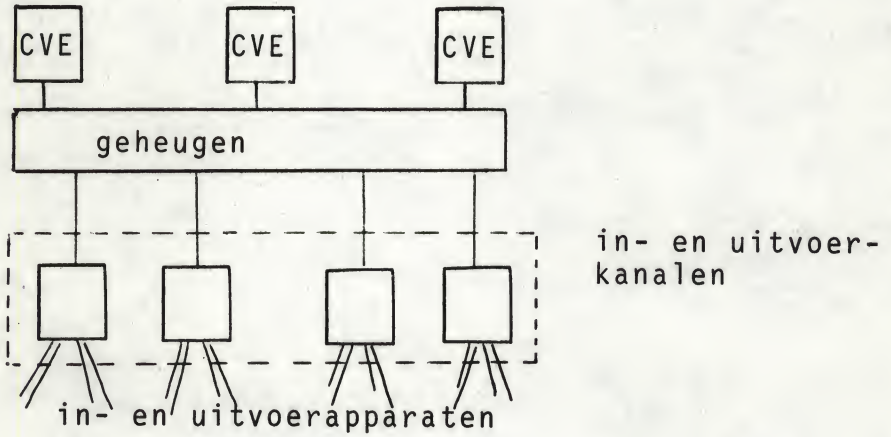
Tot nu toe hebben we steeds een machine beschouwd met één rekenorgaan, één besturingsorgaan, één geheugen alsmede een aantal in- en uitvoerorganen.
Veelal is het gebruikelijk het rekenorgaan en besturingsorgaan als één eenheid te zien, die dan wordt aangegeven met Central Proces Unit (CPE) of Centrale VerwerkingsEenheid (CVE).

Een totale computer is dan schematisch als volgt weer te geven.

Ir. D.H. Wolbers



Het geheugen geeft dus enerzijds toegang tot de CVE en anderzijds tot een aantal in- en uitvoerkanalen. Deze gedachtengang doortrekkende kunnen we het geheugen dan ook echter toegang geven tot meerdere CVE-den, waarmee we tot het volgende schema komen.



In een dergelijke configuratie wordt het nu mogelijk verschillende programma's inderdaad echt gelijktijdig te verwerken.
Het is alsof men een aantal enkelvoudige computers in elkaar geschoven heeft. Men kan het omgekeerd ook zo opvatten dat men de beschikking heeft over een aantal rekenmachines waarvan op ieder moment de samenstelling naar willekeur gewijzigd kan worden mits de totale som van de onderdelen maar gelijk blijft.
Een dergelijk systeem geven we aan met de naam multiverwerking.

Het verdient nauwelijks te worden opgemerkt dat de taak en daarmee de omvang van het coördinator programma weer aanzienlijk is toegenomen.

In vele gevallen is één CVE aangewezen voor speciaal het coördinator programma te verwerken. Deze wordt soms dan wel aangeduid met de benaming *master control*. In andere gevallen wordt op ieder moment een andere gekozen al naar gelang van de bezetting volgens prioriteiten, dit het beste uitkomt. Met de huidige stand van de software techniek vindt toepassing in deze vorm nog slechts in beperkte mate plaats.

De ontwikkeling van deze multiverwerking techniek is behalve voor een efficiënter gebruik van rekenmachines vooral van belang voor het opvoeren van de technische betrouwbaarheid.

Denkt men zich ook het geheugen opgebouwd uit een aantal equivalente eenheden dan is de gehele machine samengesteld uit een aantal onderling gelijke modulen, die elkaar niet alleen aanvullen maa zo nodig ook elkaars taak kunnen overnemen. Dit is van groot belang voor toepassingen waar een onderbreking van enige sekonden reeds funest zou kunnen zijn.
Als voorbeeld zou men kunnen denken aan het automatisch landen van vliegtuigen. Voor zulke toepassingen gebruikt men dan in feite slechts een deel van de totale capaciteit van de machine.
Treedt nu in enig onderdeel dat gebruikt wordt voor zo'n urgente toepassing een technische storing op, dan kan de machine met een minimale onderbreking toch verder werken met andere delen.
Wel gaat dit laatste gepaard met een langduriger onderbreking van een minder belangrijk programma, immers de totale capaciteit is dan kleiner geworden. Men mag natuurlijk nooit de gehele machine bezetten met alleen maar urgente programma's want dan is een continue goede werking toch weer niet verzekerd.
Daartegenover staat dan dat de delen van de machine die eigenlijk als een soort reserve dienst doen voor het urgente programma niet renteloos staan zolang de gehele machine nog goed is, maar nuttig gebruikt kunnen worden zij het dan voor minder urgente programma's.

TIME-SHARING

Nu tot slot het begrip time-sharing. Daarvoor kijken we eerst naar het gebruik van een rekenmachine. Vooral bij een geautomatiseerde administratie treffen we veelal de situatie dat relatief lange tijd achter elkaar één programma in het geheugen aanwezig is. Deze verwerkt dan misschien weliswaar veel invoergegevens enproduceert ook veel output maar programmatisch gezien blijft de toestand in de machine gedurende die tijd ongewijzigd.

In totaliteit zullen op één machine natuurlijk wel verschillende programma's gebruikt worden. Daarbij is bovendien de situatie dat niet ieder programma altijd alle mogelijkheden van de machine, speciaal wat betreft de aangesloten periferie apparatuur, nodig heeft.
Een efficiënter gebruik van een totale installatie wordt daarom bevorderd door de hiervoor beschreven werkwijzen met multiprogramming en eventueel multiverwerking.

Toch zal ook in die gevallen de programmatische indeling van de machine relatief lange tijd achter elkaar ongewijzigd blijven. Anders wordt het wanneer de programma's stuk voor stuk korter in de machine in gebruik zijn en bovendien het aantal te verwerken programma's toeneemt.
De capaciteit van de machine kan dan wel voldoende zijn maar er ontstaat de moeilijkheid dat van te voren niet meer is aan te geven welke programma's tegelijk kunnen werken en zo ja welke optimale combinaties. Een ander punt is dat men nu niet veel beter kan doen dan de verschillende programma's b.v. in de vorm van stapels ponskaarten achter elkaar aan de machine aan te bieden en deze dan principiëel in serie en waar eventueel mogelijk parallel te verwerken. Dit is een werkwijze die vooral voorkomt bij wetenschappelijke rekencentra.

Voor de individuele belanghebbende betekent dit dat hij op een gegeven ogenblik zijn programma inlevert en dan een zekere tijd moet wachten op het antwoord. Deze tijd noemt men wel de *turn around time* of rondlooptijd.Deze ligt meestal minstens in de orde enige uren helaas al te vaak ook in de orde van een dag. Gaat het de betrokkene er om een programma één keer te laten draaien om bepaalde resultaten te verkrijgen en daarmee uit, dan is een dergelijke wachttijd nog niet zo'n bezwaar. Vooral voor wetenschappelijk rekenwerk is de situatie echter geheel anders.

Men kan meestal van te voren niet precies overzien wat men hebben wil. Aan de hand van de resultaten van een eerste programma besluit men om of het programma nogmaals te gebruiken met andere invoergegevens of een andere of gewijzigde rekenmethode te gebruiken die natuurlijk weer een nieuw programma impliceert.

Een totaal onderzoek betekent dan vaak het ontwikkelen van een groot aantal programma's na elkaar, waarbij de struktuur en het beoogde doel van ieder programma bepaald wordt door de resultaten van een voorgaand exemplaar.
Daarbij komt dat ieder nieuw programma zelfs bij gebruik van makkelijke talen als FORTRAN of ALGOL, toch niet direkt foutloos is. Dit betekent dat de ontwikkeling van ieder nieuw programma tenminste een aantal "runs" op de computer betekent.

Bij een turn around time van een dag wordt een dergelijk onderzoek dan een vrij langdurige kwestie. Deze wachttijd leidt in de praktijk vaak tot een constante druk op het rekencentrum om gebruikers "aan de machine" te laten werken, zodat ze direkt resultaten zien en vlugger achter elkaar een aantal stappen in het totale proces kunnen doen. Dit is helemaal strijdig met een efficiënt gebruik van de computer.

Een poging in de goede richting is de volgende. Op één van de in- en uitvoerkanalen sluiten we een soort elektrische schrijfmachine aan en stellen die dan ter beschikking van de man die haast heeft. In de machine geven we zijn programma de hoogste prioriteit.
Via de schrijfmachine kan de programmeur nu corresponderen met de machine en zolang hij aktief is, is ook de machine als het ware aktief voor hem.
Men kan dit systeem uitbreiden door ook een tweede schrijfmachine aan te sluiten, waardoor 2 personen nu op deze wijze kunnen werken, met dien verstande dat slechts één de hoogste prioriteit kan hebben.
Bij uitbreiding met nog meer aansluitingen komt men dan echter al spoedig in moeilijkheden omdat diegenen met de laagste prioriteit dan nauwelijks meer aan bod komen.
Een ander bezwaar is dat al deze programma's gelijktijdig in het geheugen aanwezig moeten zijn hetgeen de nodige geheugenomvang dan exorbitant groot maakt.

Aan deze bezwaren kan men als volgt tegemoet komen.
Via multiplexkanaal schept men technisch de mogelijkheid een groot aantal langzaam werkende apparaten zoals teleprinters aan te sluiten. Via ieder van die teleprinters kan men in de machine werken met een eigen programma.
Deze programma's staan echter opgeslagen in een buffergeheugen, meestal een trommelgeheugen en in bepaalde gevallen een schijvengeheugen. Enkele van deze programma's zijn getransporteerd van het buffergeheugen naar het kernengeheugen. Op een gegeven moment is een van deze programma's dan aktief. Echter slechts gedurende een beperkte tijd b.v. 100 msek. Na deze periode wordt door de ingebouwde klok via de coördinator het programma stop gezet en het volgende programma krijgt een beurt. Het juist stopgezette programma wordt weer teruggetransporteerd naar het buffergeheugen.

Er vindt dus steeds een regelmatig transport plaats van programma's van en naar het buffergeheugen. Op deze wijze kan men dan wel enkele honderden aansluitingen tegelijk behandelen. Door de echte tijdsverdeling in de machine spreekt men wel van een tijdscharingssysteem. Het is natuurlijk wel duidelijk dat naarmate het aantal aansluitingen groter wordt de CVE aan steeds grotere eisen zal moeten

voldoen. Zonodig kan men dus weer combineren met multiverwerking.

Doordat men de bedieningstafels via telefoon- of telexlijnen kan aansluiten, behoeven de bedieningspunten niet meer bij de machine te zijn, maar kunnen ter plaatse, waar de gebruiker dat wenst worden aangesloten.
De gebruiker kan dus waar en wanneer hij dat wil gebruik maken van de computer. Door de vrijwel onmiddelijke responsie van de machine werkt de gebruiker in een soort samenspraak met de machine (conversational mode).
Deze wijze van werken begint thans in Amerika en ook in Europa ingang te vinden en zalzeker ook in ons land een belangrijk deel van het totale computergebruik gaan beheersen.
Dit is daarbij dan zeker niet beperkt tot wetenschappelijk gebruik. Daar speelt vooral het snel kunnen testen, veranderen en laten draaien van kleine programma's een rol. Administratief wordt van belang het ter plaatse kunnen opvragen van gegevens in allerlei vorm, het direkt kunnen aanbrengen van mutaties e.d.

Terwijl enerzijds deze tijdscharing het gebruik van computers dus nog enorm zal doen toenemen, wordt daarmee tevens het programmeerprobleem weer groter.
Er bestaat dan ook de tendens het aantal probleemgerichte talen sterk te vergroten en iedere taal meer af te richten op een bepaald vakgebied, zodat de gebruiker de machine kan toespreken zoveel mogelijk in de nomenclatuur, die hij gewend is.

GEHEUGENPROTECTIE EN DYNAMISCHE HERADRESSERING

Bij de in de vorige paragraaf genoemde systemen van multiprogramming, multiverwerking en tijdscharing doen zich ten aanzien van het geheugen enkele nieuwe problemen voor.

Doordat verschillende programma's gelijktijdig in het geheugen staan, is het niet bij voorbaat uitgesloten, dat één programma al of niet opzettelijk, gebruik maakt van de geheugenruimte van een ander programma. Nemen we eerst het niet opzettelijke geval. Het eenvoudigste voorbeeld daarvan is wel een nieuw programma, dat nog getest moet worden.

Door programmeerfouten is het zeer wel mogelijk dat een verkeerd geheugenadres van een instruktie daardoor betrekking heeft op het deel van een ander programma. Zolang deze instruktie dan maar niets op deze plaats invult heeft het andere programma daar weinig hinder van. Gebeurt dat laatste wel dan gaat er natuurlijk iets mis

zo gauw dit andere programma weer een keer aan de beurt komt en dan deze plaats nodig heeft.

Men kan zich tegen dit soort fouten beveiligen door geheugenbescherming. Dit gebeurt door bepaalde delen van het geheugen te blokkeren tegen schrijven. Er is nu echter een verschil met de uitvoering van de vroeger reeds genoemde levende en dode geheugens.
Immers iedere keer dat via de coördinator de regie overgegeven wordt aan een ander programma verandert ook de situatie t.a.v. wel en niet geblokkeerd gedeelte van het geheugen.
Het instellen van de blokkering moet dus programmatisch kunnen plaats vinden. Het geheugen wordt als het ware in stukken verdeeld over de verschillende programma's. Ook deze verdeling is weer niet vast omdat een andere serie programma's in de regel ook weer een andere verdeling vraagt. Indien men niet alleen met onopzettelijke maar ook met opzettelijke fouten rekening moet houden dan komt ook de kwestie van geheimhouding ter sprake. Immers met een bepaald programma zou men ook in het gedeelte van een ander programma kunnen "lezen". Moet men zich ook tegen dit soort fouten hoeden dan is geheugenprotectie tegen schrijven alleen niet voldoende.
Technisch zijn al dit soort wensen wel te realiseren maar kosten extra onderdelen en ontwikkeling, die men dan in de huur- of koopprijs terug vindt. Bij de aanschaffing van een machine moet men deze extra kosten ook afwegen tegen de mogelijke gevolgen van een geheugenbescherming. Geheugenbescherming zelf kan nog op verschillende manieren worden uitgevoerd. Een zeer fraaie maar ook dure oplossing is dat aan iedere geheugenplaats een aantal extra bits worden toegekend. De combinatie van deze bits is dan voor ieder programma verschillend en de doördinator geeft steeds aan het besturingsorgaan die combinatie, die betrekking heeft op het eerstvolgende te starten programma. Het voordeel van deze methode is dat men bij de verdeling van het geheugen over de verschillende programma's volkomen vrij is en dat ze zelfs door elkaar heen mogen staan. Dit laatste in tegenstelling tot een meer gebruikte oplossing waarbij het geheugen te voren in een aantal blokken is verdeeld.
Aan ieder programma kunnen nu één of meerdere blokken worden toegewezen. Deze oplossing is technisch eenvoudiger en dus goedkoper. Het nadeel is ook duidelijk. Wanneer een blok niet geheel gebruikt wordt voor een bepaald programma kan men de rest niet meer gebruiken voor een ander programma zonder de geheugenbescherming weer in gevaar te brengen. Men is dus weer minder flexibel.

Een ander geheugenprobleem is de dynamische heradressering

(dynamic relocation).Beschouwen we eerst de gevallen van multiprogramming en multiverwerking. Op een gegeven moment staan een aantal programma's in het geheugen en enkele daarvan komen klaar met hun werking. We willen natuurlijk de vrijkomende ruimte gebruiken voor één of meerdere nieuwe programma's.
Wat betreft totale omvang van het geheugen is dat misschien wel mogelijk maar door de reeds bestaande verdeling over het geheugen van de nog lopende programma's, is er misschien niet voldoende aaneensluitende ruimte te vinden.
De oplossing is dan natuurlijk de lopende programma's even stop te zetten, ze in het geheugen te verplaatsen (aaneenschuiven) om ze daarna weer verder te laten lopen. Daarbij treden echter opeens complikaties op. Immers de instrukties staan in absolute vorm in het geheugen. Verplaatsen van een programma betekent dus ook verandering van de adressen van de instrukties, maar ook weer niet van allen (denk in SERA aan instrukties met een ster) Bovendien getallen en constanten mogen natuurlijk niet gewijzigd worden. Ernstig is dan ook dat juist op het moment van onderbreking in het rekenorgaan misschien een adresrekening plaats vindt (denk aan voor de voet schrijven). Dat zou na verplaatsing dus ook gewijzigd moeten worden.
Dit probleem is voor een machine zoals b.v. SERA is uitgevoerd dan ook onoplosbaar, tenzij men de techniek van de machine wijzigt. Deze bestaat dan hierin dat een programma niet meer met absolute adressen in de instrukties in het geheugen gezet wordt, maar met relatieve t.o.v. het begin van het programma. In het besturingsorgaan bevindt zich dan een speciaal register soms basisregister genoemd. Dit basisregister bevat het absolute beginadres voor het programma dat op dat moment aktief is. Iedere instruktie met relatief adres wordt dan pas op het moment van uitvoering omgezet in een instruktie met absolute adres. Gaat de regie over op een ander programma dan wordt ook de inhoud van het basisregister gewijzigd. Dit zelfde geldt ook wanneer het programma in het geheugen verplaatst wordt.

Meestal vormt het gebruik van een dergelijk basisregister ook een deel van de geheugenprotectie. Het is duidelijk dat een dergelijk probleem van dynamische heradressering zich ook voordoet bij gebruik van tijdscharing, omdat het niet wel mogelijk is een zelfde programma steeds iedere keer op dezelfde plaats in het geheugen te zetten.

Welke wijze verder ook gevolgd wordt, in ieder geval zullen in de machine speciale instrukties ingebouwd moeten zijn om de geheugenbescherming te kunnen instellen en wijzigen en dus te kunnen aanpassen aan de momentane behoefte. Dit werk is dus tevens weer een taak voor de coördinator, die daarvoor dan gebruik kan maken van deze

speciale instrukties.
Bij dit laatste moeten we tevens opmerken dat het coördinatorprogramma deze instrukties niet alleen kan gebruiken, maar, met uitsluiting van de andere programma's, ook alleen maar mag gebruiken.
Immers wanneer een gewoonprogramma deze instrukties zou kunnen gebruiken, dan zou dit programma de beschermingssleutel kunnen veranderen, waarmee de bescherming natuurlijk is opgeheven. Dit is als het afsluiten van een brandkast en dan de sleutel in het slot laten zitten.
Deze instrukties behoren dan ook tot de groep beschermde instrukties.